精通 Elastic Stack



[印]Y.古普塔（Yuvraj Gupta）
[印]R.K.古普塔（Ravi Kumar Gupta） 著
高凯 岳重阳 苗雪立 张思琪 译

清华大学出版社

精通 Elastic Stack

Yuvraj Gupta　Ravi Kumar Gupta 著

高　凯　岳重阳　苗雪立　张思琪　译

清 华 大 学 出 版 社
北　京

北京市版权局著作权合同登记号　图字：01-2017-6579
Yuvraj Gupta and Ravi Kumar Gupta
Mastering Elastic Stack
ISBN-13：978-1786460011

图书在版编目（CIP）数据

精通 Elastic Stack/（印）Y. 古普塔（Yuvraj Gupta），（印）R. K. 古普塔（Ravi Kumar Gupta）著；高凯，等译. —北京：清华大学出版社，2018
书名原文：Mastering Elastic Stack
ISBN 978-7-302-49243-6

Ⅰ. ①精…　Ⅱ. ①Y…　②R…　③高…　Ⅲ. ①数据处理　Ⅳ. ①TP274

中国版本图书馆 CIP 数据核字（2017）第 329710 号

责任编辑：焦　虹
封面设计：傅瑞学
责任校对：时翠兰
责任印制：沈　露

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：北京嘉实印刷有限公司
经　　销：全国新华书店
开　　本：185mm×240mm　　**印　　张：**26.75　　**字　　数：**544 千字
版　　次：2018 年 8 月第 1 版　　**印　　次：**2018 年 8 月第 1 次印刷
定　　价：79.90 元

产品编号：076709-01

2017 Packt 出版社版权声明

首次出版：2017 年 2 月
产品基准码：1240217

出版商：Packt Publishing Ltd.
地址：Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.
www.packtpub.com

荣誉及致谢

作者

Ravi Kumar Gupta

Yuvraj Gupta

审校人

Israel Farfan

Marcelo Ochoa

组稿编辑

Veena Pagare

采稿编辑

Tushar Gupta

内容编辑

Narendrakumar Tripathi

技术编辑

Sachit Bedi

文字编辑

Safis Editing

项目协调人

Devanshi Doshi

校对

Safis Editing

索引

Aishwarya Gangawane

图表

Disha Haria

产品协调

Deepika Naik

译　者　序

在大数据时代，建立一个网站或应用程序，搜索、挖掘与分析功能是必备的。本书从分布式大数据搜索、日志挖掘、可视化、数据监控与管理等多个角度出发，在 Elastic Stack 5 的基础上，介绍了 Elasticsearch、Logstash、Kibana、Beats、X-Pack 等诸多相关组件。原著作者 Ravi Kumar Gupta 除作为计算机专业技术书籍的审稿人外，也是开源软件社区的维护者；原著的另一名作者 Yuvraj Gupta 是大数据实践领域的技术顾问，其研究领域涉及大数据、数据分析、数据可视化和云计算等。二人合作完成的这部著作，从面向实践的角度出发，比较全面地介绍了 Elastic Stack 5 的实际应用；并结合一些项目实例，介绍了大数据分析的部分关键技术。我们认为，无论对初学者还是有经验的开发人员，原著都是很有参考价值的。它不仅内容全面，而且表达比较通俗易懂，实践指导性较强。原著强调实践、面向初学者，通过实战讲解的方式，可让读者更好地了解相关组件的应用。通过翻译这本书，我们也从中收获很多、受益颇丰。

本书由高凯、岳重阳、苗雪立、张思琪合作翻译。其中，高凯完成了第 1、3、7 章，岳重阳完成了第 5、9、10、11、12 章，苗雪立完成了第 4、6 章，张思琪完成了第 2、8 章。最后，由高凯统稿。本书翻译过程中得到了多方面的支持与帮助；何晓艺、张姗姗、孟天宏、刘多星、高成亮、毛雨欣、聂颖杰、韩佳、谢宇翔、李明奇、侯雪飞、杨聪聪、江跃华等均提供了协助。

尽管本书的译者在大数据搜索与挖掘及可视化管理方面有一定的经验，也出版过相关的著作、教材等，但毕竟水平有限，译文中难免有不足和有待商榷之处，敬请读者批评指正。

译　者

关于本书作者

本书作者 Ravi Kumar Gupta 是计算机专业技术书籍审稿人、开源软件社区维护者。他于印度彼拉尼邦的伯拉科技学院(Birla Institute of Technology and Science，BITS)获得软件系统硕士学位，于印度拉贾斯坦邦斋浦尔(Jaipur)的 LNMIIT(The LNM Institute of Information Technology)获学士学位，在技术上擅长门户网站研发。

他目前就职于 Azilen Technologies 公司，任技术架构师和项目经理，曾担任 CIGNEX Datamatics 的首席技术顾问[①]。他曾是开源组织 TCS[②] 的核心成员，从事开源软件社区管理和其他的用户界面技术研发。在其职业生涯中，他致力于使用最新技术构建企业级解决方案，并注重用户界面和开源工具的使用。

他喜爱写作、学习，热衷于讨论 IT 新技术。大学期间他的研究领域涉及基于爬虫的搜索引擎的研发，是技术爱好者。他也是由 Packt 出版社出版的 *Test-Driven JavaScript Development* 的作者之一，是软件社区论坛的活跃成员。他目前维护其博客 http://techdc.blogspot.in，并经常在上面发表计算机相关技术的系列文章。

他还维护着 TCS 和 CIGNEX 软件社区(Liferay 5.x 和 6.x 版本)，同时也是 Packt 出版社出版的 *Learning Bootstrap* 的审校人。其联系方式如下：

Skype：kravigupta；Twitter：@kravigupta；LinkedIn：https://in.linkedin.com/in/kravigupta

“感谢我的妻子 Kriti。正是她的鼓励和支持，伴我度过了本书写作的艰辛时光。感谢我的妻子、我的家庭，特别是我的岳父母，他们为我提供了很多帮助。更要感谢本书合著者 Yuvraj。作为好朋友，他为我提供了很好的支持和理解，没有他，本书是不可能完成的。我还要感谢 Packt 出版社、审校人、编辑团队的同事们。感谢你们！谢谢！”

本书的另一位作者 Yuvraj Gupta 的研究领域涉及大数据、数据分析、数据可视化和云计算。他是大数据实践领域的技术顾问，喜爱各种社交平台，是小工具插件的开发与爱好者。他喜爱美食、运动、各种影视剧，也一直跟踪最近技术发展动态。他在 Packt 出版社出版了 *Kibana Essentials*。其联系方式如下：

E-mail：gupta.yuvraj@gmail.com；LinkedIn：www.linkedin.com/in/guptayuvraj

---

① 译者注：http://www.cignex.com/。

② 译者注：https://sourceforge.net/projects/opentcs/files/?source=navbar。

“本书写作过程中，感谢我的家人和朋友提供的支持和鼓励。感谢本书审校人以及 Packt 出版社为本书做出奉献的全体同事，没有你们的帮助，本书是不可能出版的。感谢所有直接或间接为我写作提供支持的同事。同时，感谢我的老师、教授、导师、学院、大学，你们培育并传授给我知识。感谢我的合作者 Ravi。没有这些帮助、信任和支持，本书是不可能完成的。”

关于本书审校人

Marcelo Ochoa 就职于 Universidad Nacional del Centro de la Provincia de Buenos Aires 系统实验室，是 Scotas. com（这是一家使用 Apache Solr 和 Oracle 的实时搜索公司）的 CTO。平时，除完成大学中的工作，他也从事和 Oracle 及大数据技术相关的工作，做过一些和 Oracle 相关的工作（如 Oracle 手册的翻译、CBT 多媒体等），其技术背景为数据库、网络、互联网、Java 技术等。在 XML 领域，他以 Apache Cocoon 工程的 DB Generator 的研发者而著称。他也在开源项目 DBPrism 和 DBPrism CMS（这是一个由 Oracle JVM Directory 实现的基于 Lucene-Oracle 的集成项目）中提供服务，在网站 https://restlet.com/project 有工作经历。他在项目中主要从事 Oracle XDB Restlet Adapter 的开发工作，这是 Oracle JVM 中专注于 REST 的 Web 服务的一个替代品。从 2006 年起，他成为 Oracle ACE program 中的一员。Oracle ACE program 在 Oracle 社区享有崇高声望，拥有众多热情的支持者和倡导者。他也是 ACES 在 Oracle 技术和应用社区的倡导者。作为合著者，他参与编写了由 Digital 出版社出版的 *Oracle Database Programming using Java and Web Services* 和由 Wrox 出版社出版的 *Professional XML Databases*。他还是由 Packt 出版社出版的几部技术著作，如 *Apache Solr 4 Cookbook*，*ElasticSearch Server* 等的审校人。

www. packtpub. com

要获取本书相关程序文件，可登录 www. packtpub. com。

你知道吗？Packt 出版社为出版的每一部书籍提供 PDF、ePub 格式的电子书。可以登录 www. packtpub. com 更新、下载电子书，并可享受电子书折扣。如果需要，请联系我们 service@packtpub. com。

在 www. packtpub. com，也能阅读一些免费的技术资料。注册成为自由的信息提供者，将会获得更高折扣并获得由 Packt 出版社提供的纸质书和电子书。



https://www. packtpub. com/mapt

访问 Mapt 网站，可获取更多相关内容。该网站提供所有 Packt 出版社的书籍、视频课程和在工业界领先的工具软件等。这些能帮助你规划未来的个人发展，并获得职业提升的机会。

可以通过浏览器访问该网站，复制、粘贴、打印、收藏由 Packt 出版社发行的电子书。

致 消 费 者

感谢购买由 Packt 出版社出版的图书。在 Packt 出版社，保证图书质量是一切编辑活动的核心。为了帮助我们提高图书质量，诚邀您访问本书在亚马逊的网站：https://www.amazon.com/dp/1786460017，并留下您宝贵的意见及建议。

如果有意成为审稿人，可发送电子邮件到 customerreviews@packtpub.com。为表示感谢，将会赠送您免费的电子书和视频资料。

让我们携手努力，共同提高图书质量！

Foreword



前言

如果数据不能有助于做出决定、提升目前的系统性能，即使是结构化的数据，那也是无用的（更何况非结构化的数据了）。如果对数据感兴趣，或者需要处理用户各种类型的日志数据，或者需要设计高可扩展性的分析系统，或者需要管理日志并进行实时的数据分析，本书可提供“一站式”解决方案。通过集成 Elasticsearch、Logstash、Beats、Kibana 等多个流行软件，ELK Stack 已经进化为 Elastic Stack，它能以近乎实时的高效处理方式，处理几乎各种类型的结构化和非结构化的数据。

本书首先介绍有关 Elastic Stack 的基础知识，之后会涉及一些更复杂和高级的内容。我们将帮助你借助 Elastic Stack，应对数据分析的挑战，并以内网应用环境为例，带你从实战角度理解 Elastic Stack 组件的使用。通过学习，你将会了解日志分析和可视化的高级技术。另外，也将以实例的方式介绍一些新特性——如 Beats 和 X-Pack。

最终，你将会看到如何使用 Elastic Stack 解决现实世界中的实际问题。同时，本书也将会介绍一些在应用 Elastic Stack 中需要注意的问题。

本书包含哪些内容？

第 1 章，Elastic Stack 概述。通过搭建 Elastic Stack 的各种组件，介绍从 ELK 到 Elastic Stack 的转化。

第 2 章，使用 Elasticsearch。介绍如何在工程项目中开始使用 Elasticsearch，介绍 Elasticsearch 工作机制，并介绍各种 Elasticsearch API 和聚合 Aggregations 的用法。

第 3 章，Logstash 及其插件的使用。内容涉及 Logstash 简介、Logstash 结构、各种插件的用法示例。最后，介绍一个有关 Logstash 配置文件以及日志解析的实例。

第 4 章，Kibana 界面设计。介绍各种 Kibana 界面的用法，并通过一些例子来演示如何将各种界面相结合并设计面板可视化。

第 5 章，使用 Beats。介绍 Beats，讲述 Beats 和 Logstash 的不同之处，

并介绍各种类型的 Beats 的功能以及设置方法，最后介绍在 Elastic Stack 中如何使用 Beats。

第 6 章，Elastic Stack 实战。介绍在局域网环境下实际使用 Elastic Stack 的方法，并通过例子解释如何使用 Elastic Stack 组件解决实际的具体问题。

第 7 章，个性化配置 Elastic Stack。介绍如何扩展 Elastic Stack 中的各种组件，并介绍定制个性化组件的方法。

第 8 章，Elasticsearch API。通过介绍各种 Elasticsearch API 的用法，使读者理解 Elasticsearch 诸模块的工作机制；并介绍节点、组件发现策略及使用 Java 客户端实现对 Elasticsearch 的各种操作。

第 9 章，X-Pack 插件中的 Security 与 Monitoring 组件。内容涉及 X-Pack 简介与安装、安全和监控等，本章还涉及 Shield、Marvel、Profiler 相关功能的使用。

第 10 章，X-Pack 插件中的 Alerting、Graph 和 Reporting 组件。本章还介绍了 Watcher、Graph、Reporting 等组件的使用、功能、特性等。

第 11 章，最佳实践范例。本章将分成多个小节，使读者理解为什么需要遵循最佳实践标准。

第 12 章，案例分析——Meetup。通过扩展 Logstash 的功能、生成新的功能插件等方法，使读者加深对相关问题的理解，学习如何应用 Elastic Stack 组件来分析和处理端到端的 Meetup 数据，展示 Elastic Stack 在数据分析方面的强大功能。

需要下载什么软件工具?

为了能运行书中的示例，下表列出了可能需要的软件和工具。通过表中列出的链接，可下载相应章节需要的软件。

软　件	版本	链　　接
Elasticsearch	5.1.1	https://www.elastic.co/downloads/past-releases/elasticsearch-5-1-1
Logstash	5.1.1	https://www.elastic.co/downloads/past-releases/logstash-5-1-1
Kibana	5.1.1	https://www.elastic.co/downloads/past-releases/kibana-5-1-1
Filebeat	5.1.1	https://www.elastic.co/downloads/past-releases/filebeat-5-1-1
Packetbeat	5.1.1	https://www.elastic.co/downloads/past-releases/packetbeat-5-1-1
Winlogbeat	5.1.1	https://www.elastic.co/downloads/past-releases/winlogbeat-5-1-1
Metricbeat	5.1.1	https://www.elastic.co/downloads/past-releases/metricbeat-5-1-1
Elasticsearch	1.4.1	https://www.elastic.co/downloads/past-releases/elasticsearch-1-4-0

续表

软　件	版本	链　　接
Liferay	6.2CEGA4	https://sourceforge.net/projects/lportal/files/Liferay Portal/6.2.3 GA4/liferay-portal-tomcat-6.2-ce-ga4-20150416163831865.zip/download[①]
Java	8.x	http://www.oracle.com/technetwork/java/javase/downloads/index.html
Elasticray	1.2.0	https://web.liferay.com/marketplace/-/mp/application/41044606
Go	1.7.5	https://golang.org/dl
Ruby	2.4.0	https://www.ruby-lang.org/en
NodeJS	6.9.0	https://nodejs.org/en/download/releases/
Gradle	2.13	https://gradle.org/gradle-download
Python	2.7.10	https://www.python.org
Virtualenv		https://virtualenv.pypa.io/en/stable/
cookiecutter		https://github.com/audreyr/cookiecutter

谁适合阅读本书?

如果曾经听说过 ELK Stack,想学习有关它的最新发展,了解它如何演化成为 Elastic Stack,那么这本书就是为你而准备的;如果正在进行数据分析,或计划通过可视化技术来展现数据,本书也适合你。它能帮助你了解 Elastic Stack 中的组件是如何发挥作用的。

惯用法与记号说明

为区分不同种类的信息,本书采用了不同风格的文本。这里列出一些例子并进行说明。

文本中的代码、数据表名字、文件夹名、文件名、文件扩展名、路径名、虚拟的 URL、用户输入、Twitter 句柄等以如下方式展示。例如,下面这些代码读取数据并赋值给 BeautifulSoup 函数。

```
#import packages into the project
from bs4 import BeautifulSoup
from urllib.request import urlopen
import pandas as pd
```

当需要提醒注意某部分代码时,以黑体显示。

```
<head>
```

① 译者注:原文中,该 URL 中的空格是用%20 表示的。这里直接以空格表示。

```
<script src="d3.js" charset="utf-8"></script>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width">
<title>JS Bin</title>
</head>
```

基于命令行的输入或输出，将按如下方式展现：

```
C:\Python34\Scripts>pip install -upgrade pip
C:\Python34\Scripts>pip install pandas
```

新词和重要词汇以黑体显示。在屏幕、菜单或对话框中显示的内容，将采用如下方式展示。例如，为了下载新的模块，将在 **Files** | **Settings** | **Project Name** | **Project Interpreter** 菜单选项中完成相应的操作。

> 警告或重要的说明将出现在这种框中。

> 提示和操作技巧将出现在这种框中。

读者意见反馈

欢迎读者提出反馈意见。请告诉我们，针对本书有什么意见？喜欢什么？不喜欢什么？读者的意见反馈对作者来说是很重要的，可便于我们进一步修改、完善内容。可以发送电子邮件到 feedback@packtpub.com。请在邮件主题中注明本书书名。如果您是某一个领域的专家并且愿意撰写相关内容的图书，请参阅 Packt 出版社如下网站中的“作者须知”：www.packtpub.com/authors。

读者服务

作为 Packt 出版社尊贵的读者，从购买本书开始，您将享受 Packt 出版社提供的各种服务。

下载示例代码

可以在 Packt 出版社提供的 http://www.packtpub.com 网站，用您的账号下载本书的示例代码。不论在哪里购书，均可访问 http://www.packtpub.com/support。完成注册后，我们会通过电子邮件给您发送相关文件。

按照如下步骤下载代码文件：

(1) 用您的电子邮件地址和设定的密码访问我们的网站，完成用户登录或新用户注册。

(2) 定位到页面顶部的 SUPPORT 标签页。

（3）单击 Code Downloads & Errata。
（4）在搜索框中输入书名。
（5）选择要下载代码文件的书名。
（6）从下拉菜单中选择所购买的图书。
（7）单击 Code Download。
完成下载后，确保计算机中有如下的新版解压缩软件。

- Windows 环境：WinRAR/7-Zip；
- Mac 环境：Zipeg/iZip/UnRarX；
- Linux 环境：7-Zip/PeaZip。

本书提供的代码文件也可从 Github 下载：https://github.com/PacktPublishing/Mastering-Elastic-Stack。其他更丰富的相关资源和视频也可访问：https://github.com/PacktPublishing/。去试试吧！

勘误表

虽然我们已经尽力确保书中内容的准确性，但错误可能仍无法完全避免。如果在书中文字或代码中发现了错误，请告诉我们，我们将不胜感激。这不仅能避免错误给读者带来疑惑，也能帮助我们提高图书再版的质量。如果发现书中有错误，请访问 http://www.packtpub.com/submit-errata，选择该书，单击 Errata Submission Form 链接，输入有关勘误的信息。一旦确认，提交的勘误信息将会更新到我们的网站，或者追加到现有的勘误表中。

要查阅早期提交的勘误信息，可访问 https://www.packtpub.com/books/content/support。在搜索框中输入书名，会出现相关的勘误信息。

著作权

在各种媒体上，通过互联网对著作权侵权是目前经常发生的问题。在 Packt 出版社，我们非常注重对著作版权的保护。如果在互联网上见到对我们作品的非法获取、复制等各类盗版行为，请提供给我们有关的地址、网站等信息，以便进行处理。

请联系我们：copyright@packtpub.com，并请提供疑似盗版的相关链接。

非常感谢您在保护著作权方面为我们提供的帮助，以便我们能为您带来更多有价值的内容。

其他问题

如果您有关于本书的其他任何问题，可以通过如下电子邮件联系我们 questions@packtpub.com。我们将竭诚为您提供帮助。

Contents

目录

Elastic Stack 概述

人们很容易读取一个几兆到几百兆的日志文件，因为数据库完全有能力处理这种体量的数据或文件。然而，如今动辄就需要处理几个"T"、几个"P"级别的数据，这种大数据的发展是大势所趋且在未来发展速度很快。在大数据需求的推动下，常规的文本编辑器或字处理软件工具往往难以应对（它们往往不能处理这样的大数据集），但对原始大数据的分析，能提高我们对数据的洞察力。我们需要对大数据进行有效处理的工具。接下来，将会学习如何对海量日志文件进行管理，或以某种合适的方式对海量数据进行索引、分析。使用Google，能搜索到 ELK Stack[①]——**E**lasticsearch 可以管理数据，**L**ogstash 可以读取不同来源的日志数据，**K**ibana 能够可视化这些数据。

最近，ELK 已经演化为 Elastic Stack。本章将带你了解并搭建 Elastic Stack 环境。本章主要内容如下：

- ELK Stack 简介；
- Elastic Stack 的诞生；
- 哪些组织在用 Stack?
- Stack 的组成；
- 设置 Elastic Stack；
- 了解 X-Pack。

1.1 ELK Stack 简介

ELK Stack 起源于 Shay Banon 以 **Compass** 为基础研发的如今名为 **Elasticsearch** 的开源工程工具。现在，Elasticsearch 已经成为一款非常流行的开源的数据库搜索引擎产品。从这之后，在分布式工作模型的基础上研发出了 **Kibana**，它可用于可视化 Elasticsearch 中索引的数据。在这之前，我们往往使用 **Rivers**（它提供了一种特殊的数据输入机制），把数据

① 译者注：ELK 分别源自 Elasticsearch、Logstash、Kibana 的首字母。

同步到 Elasticsearch 中。

然而，随着这种方法的应用日益广泛，我们需要一个工具。借助这个工具，可灵活地把数据插入到 Elasticsearch 中，而且能便利地执行各种数据转换操作，如对非结构化数据的结构化处理，完成对数据处理的有效控制。在这种需求前提下，**Logstash** 诞生了。它也被归并到 Stack 中。至此，Elasticsearch、Logstash、Kibana 这三个工具统称为 **ELK Stack**。

下图是基于 ELK Stack 的数据处理流程。



从上图中我们可以看出，数据是由 Logstash 读取的，然后在 Elasticsearch 中完成索引。最后，我们能利用 Kibana，从 Elasticsearch 中读取索引数据并以图表形式可视化数据。下面让我们来分别理解这些组件，以及它们在 Stack 中所起的作用。

1.1.1 Logstash

前面已经提到，在 ELK Stack 出现之前，Rivers 被用于将数据同步到 Elasticsearch 中。对于 ELK Stack 来说，Logstash 是所有类型的数据的入口点。相比于 Rivers，Logstash 拥有非常多的 Input 输入数据类型的插件，这些 Input 插件可用于从大量不同来源的信息中读取数据；同时，它也拥有非常多的 Output 输出数据类型插件，而这些 Output 插件可用于把数据提交到各种不同的目的地——其中的一种插件就是用于把数据传输到 Elasticsearch 中去。

Logstash 流行之后，由于集群稳定以及性能等方面的原因，Logstash 最终取代了 Rivers 插件。

Logstash 不仅用于简单地把数据从一个地方传送到另一个地方。它能帮助我们搜集原始数据，修改/过滤数据并将其转换成某种有含义的数据，完成数据的格式化和重新组织数据等。被 Logstash 加工后的数据可提交给 Elasticsearch 进行处理。如果没有其他可用的插件来完成从某种特殊的数据源中读取数据，将数据写到某种目的地，按照你自己设定好的方式修改数据等，那么使用 Logstash 可以完成上述功能，且可足够灵活地重写相应的数据处理插件。

简言之，Logstash 是一个高灵活度的、具有丰富插件的、能从所选择的某个位置源读取数据的一种开源工具软件。它能按所定义的配置信息来规范化数据，并根据需要将其发送到指定的目的地。

我们将会在第 3 章中学习 Logstash 及其组件，在第 7 章中学习个性化定制 Elastic Stack 等内容。

1.1.2 Elasticsearch

Logstash 读取的数据可输出到 Elasticsearch 中，完成数据的索引。Elasticsearch 不仅用于数据的索引，还是一个全文检索的搜索引擎，具有高可扩展性，也能提供分布式系统的很多功能。Elasticsearch 以索引的形式管理并维护数据，并通过相应的 API 实现对相关数据的查询、聚合分析等服务。Elasticsearch 构建在 **Lucene** 上，也能提供很多 Lucene 所具有的功能。

我们将在第 2 章走进 Elasticsearch、第 7 章个性化定制 Elastic Stack、第 8 章 Elasticsearch API 中学习更多关于 Elasticsearch 的内容。

1.1.3 Kibana

Kibana 使用 Elasticsearch 提供的 API 来读取/检索存放在 Elasticsearch 中的索引数据，并以图表等形式对这些数据进行可视化分析。Kibana 是一种 Web 应用程序，能提供高度可配置的用户接口，能查询数据，生成大量用于可视化的图表及分析存储的数据。

我们将在第 4 章学习 Kibana 界面。

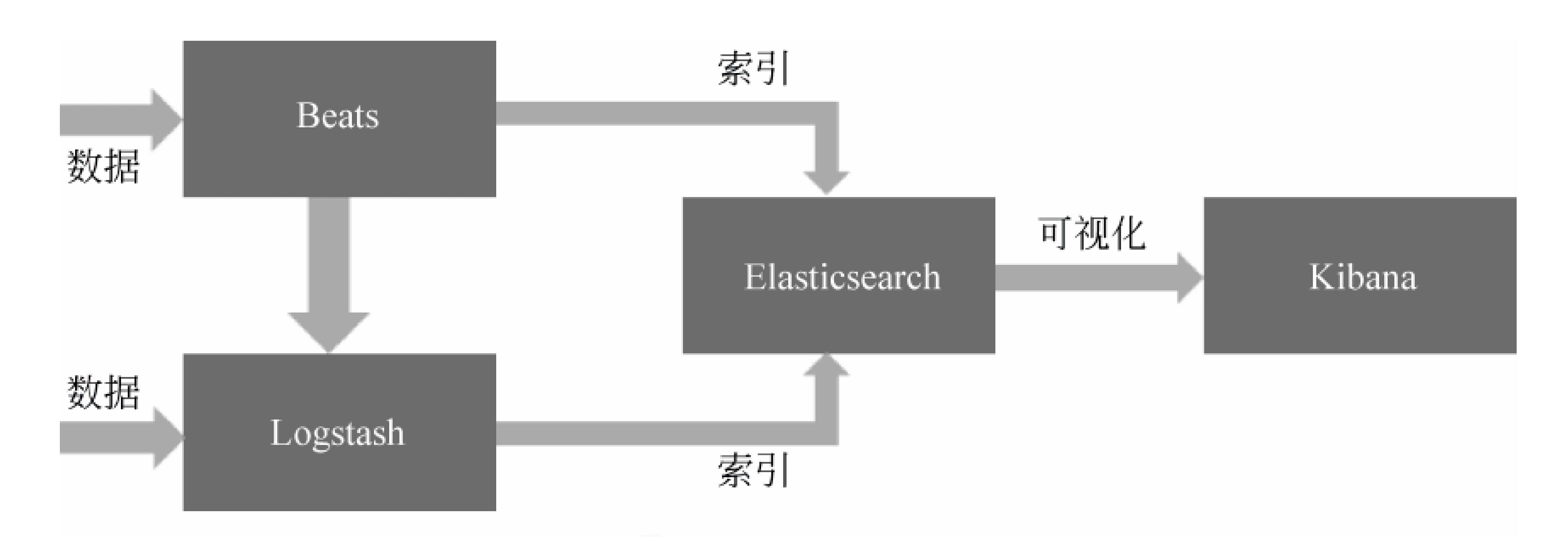
在上述性能强劲的 ELK Stack 出现之后，随着时间的推移，一些重要且复杂的现实需求也出现了，比如认证(authentication)、安全(security)、通知(notification)等。这种现实需求的不断增长也导致了一些其他工具的陆续出现，如 **Watcher**(如果数据发生改变，它能提供警告和通知)、**Shield**(提供安全集群的认证、授权等)、**Marvel**(监控集群的统计特征)、**ES-Hadoop**、**Curator**、**Graph** 等。

1.2 Elastic Stack 的诞生

起初，所有读取数据的工作都是由 Logstash 完成的，但是这是一种资源消耗，因为 Logstash 需要运行在 Java 虚拟机上，它会消耗大量内存。软件研发社区认为需要提高其性能，并使用管道(pipeline)处理机制——一种友好且轻量级的方式来处理资源。诞生于 2015 年的 **Packetbeat** 工程能有效分析使用不同网络协议的数据包，解析数据并将其输出至 Elasticsearch。这是一种很自然的轻量级研发理念，由此一种新概念(即 Beats)诞生了。**Beats** 是由 Go 编程语言编写的。现在的 ELK 已不仅包括之前的 Elasticsearch、Logstash、Kibana 成员，Beats 也加入到 ELK Stack 家族中，并成为其中的一个重要组件。

数据处理管道的示意图如下图所示。从该图中可以看出，Beats 用于读取、解析并将数据输出到 Elasticsearch 或 Logstash 中。不同的是，它是一种轻量级的、服务于某种特殊用途的代理 Agents(它可以是 Metricbeat、Filebeat、Packetbeat 等)。它们都是由 Elastic 开发

团队提供的。除此之外,在开发社区中也有为数众多的其他 Beats。如果有某种特殊需求,就可以通过使用 libbeat 库,自行开发满足个性化需求的 Beats。



简言之,Beats 能以一种轻量代理 Agents 的方式,把数据输出到 Logstash 或者 Elasticsearch 中。如果需要,还可以通过使用 libbeat 库的方式,订制满足个性化需求的专属 Beats。

我们将在第 5 章中学习使用 Beats 的知识。

从最初被称为 ELK Stack 的 Elasticsearch、Logstash、Kibana,到 Beats 加入大家族后改称为 Elastic Stack,其实 Elastic Stack 涵盖的内容还不止这些。Elastic Stack 的起始版本号是 5.0.0,这个版本同时适用于 Elastic Stack 中的所有组件。

该版本及其发行的方法不仅适用于 Elastic Stack,对 Elastic 家族中的其他工具也有效。但由于工具众多,所以在软件一致性方面存在协调性问题。因为每个工具都有其自己的版本号,并且可能由于各个版本之间的不兼容而导致一些问题的出现,所以为解决上述问题,所有工具将一并构建、测试和发布。

在产生数据管道的过程中,所有这些组件都发挥了重要作用:Beats 和 Logstash 用于搜集、解析、传输数据;Elasticsearch 负责对数据的索引①,而这些索引最后会被 Kibana 用于数据的可视化。在基于 Elastic Stack 的数据处理管道中,其他工具则用于增加系统安全性,提供通知,完成性能监控,完成设置等。

1.3 谁在使用 Elastic Stack?

在过去的几年间,基于 Elastic Stack 的应用在快速增长。本节将介绍 Elastic Stack 的应用案例,帮助读者了解 Elastic Stack 如何助力软件的研发。

① 译者注:Elasticsearch 还负责优化索引,完成对索引数据的检索。

1.3.1　Salesforce

Salesforce 研发了一个名为 **ELF**(**Event Log Files**)的新插件，用于搜集 Salesforce 的日志数据，审计用户的实际数据，其目的在于分析数据来理解在 Salesforce 中的用户行为，并预测其未来发展趋势。

这个插件可以在 GitHub 中获取：

```
https://github.com/developerforce/elf_elk_docker
```

ELF 是 **Event Log Files** 的缩写。该插件简化了 Stack 架构的配置，允许下载事件日志文件(Event Log Files)并得到索引，最终通过可视化的 Kibana 来分析数据。该应用使用了 Elasticsearch、Logstash 和 Kibana。

1.3.2　CERN

在**欧洲原子能研究组织**(**CERN**)中，Elastic Stack 完成的工作任务不是一项而是五项。Elastic Stack 主要用于完成如下工作任务：

- 消息管理(Messaging)；
- 数据监控(Data monitoring)；
- 云基准服务(Cloud benchmarking)；
- 系统架构监控(Infrastructure monitoring)；
- 作业调度(Job monitoring)。

多种 Kibana 面板在这里被用于对数据的可视化工作。

1.3.3　Green Man Gaming

Green Man Gaming 是一款在线游戏平台。在这个平台上，游戏开发者发布其研发的游戏产品，目的在于为玩家提供独特的、良好的游戏体验。他们也开始使用 Elastic Stack 来传输日志分析、完成搜索和分析游戏数据。

他们从设置 Kibana 面板开始，获取了有关玩家的数量、国籍和所使用的货币等信息。这将帮助他们更好地理解用户需求，改进和提升服务响应质量。

除了这些，Elastic Stack 也被其他很多公司用于分析相应的数据。有时候，并非所有的组件都能被派上用场，例如 Beats 和 Logstash 等不是每次都能派上用场。有时候，可能只是结合使用 Elasticsearch 和 Kibana。

如果我们看一下该组织的用户，就会发现那些热衷于大数据分析、商业智能、数据可视化、日志分析、数据科学处理与应用等方面的人往往会愿意使用这个工具。

1.4 竞争者

Elastic Stack 本来就不缺竞争对手，因为它一出现就成为近年来市场上其他工具的强有力的竞争对手，并且它还一直在发展。下面是一些和 Elastic Stack 类似的软件产品：

- 开源软件：
  - **Graylog**：可访问 https://www.graylog.org/。
  - **InfluxDB**：可访问 https://influxdata.com。
- 其他：
  - **Logscape**：可访问 http://logscape.com。
  - **Logscene**：可访问 http://sematext.com/logscene/。
  - **Splunk**：可访问 http://www.splunk.com。
  - **Sumo Logic**：可访问 https://www.sumologic.com。
- Kibana 的竞争者：
  - **Grafana**：可访问 http://grafana.org。
  - **Graphite**：可访问 https://graphiteapp.org。
- Elasticsearch 的竞争者：
  - **Lucene/Solr**：可访问 https://lucene.apache.org 或 http://lucene.apache.org/solr。
  - **Sphinx**：可访问 http://sphinxsearch.com。

上面大多数产品也做日志管理，但 Elastic Stack 已经不局限于此了。它提供了处理各种类型数据的功能，而非仅仅局限于日志管理。

1.5 设置 Elastic Stack 的使用环境

从本节开始，我们将要在两种流行的操作系统 Windows 和 Ubuntu 上安装 Elastic Stack 的 4 个组件。在安装 Elasticsearch 和 Logstash 之前，需要先安装好 Java 环境。如果已经在计算机上安装好 Java 环境，可略过这一部分。

1.5.1 安装 Java

本节中，计算机中需要安装 JDK，用于访问到 Elasticsearch。我们应该安装好 Oracle Java 8(Oracle JDK1.8.0_73)版本的 Java 运行环境，它也是 Elasticsearch 5.0.0 推荐使用的 Java 版本。

1.5.1.1 在 Ubuntu 14.04 上安装 Java

按如下步骤，可通过终端和 APT 包[1]的方式安装 Java 8。

(1) 执行如下命令，添加 Oracle Java PPA 仓库(Personal Package Archive)[2]获取 APT 列表：

```
sudo add-apt-repository -y ppa:webupd8team/java
```

> 这里使用了第三方资源。这不违反 Oracle Java 规范。PPA 直接从 Oracle 下载并安装 Java。

sudo 命令执行后[3]，如果不是 root 用户，则需要输入密码；成功后，可能会显示"OK"，表明存储内容已被正确导入。

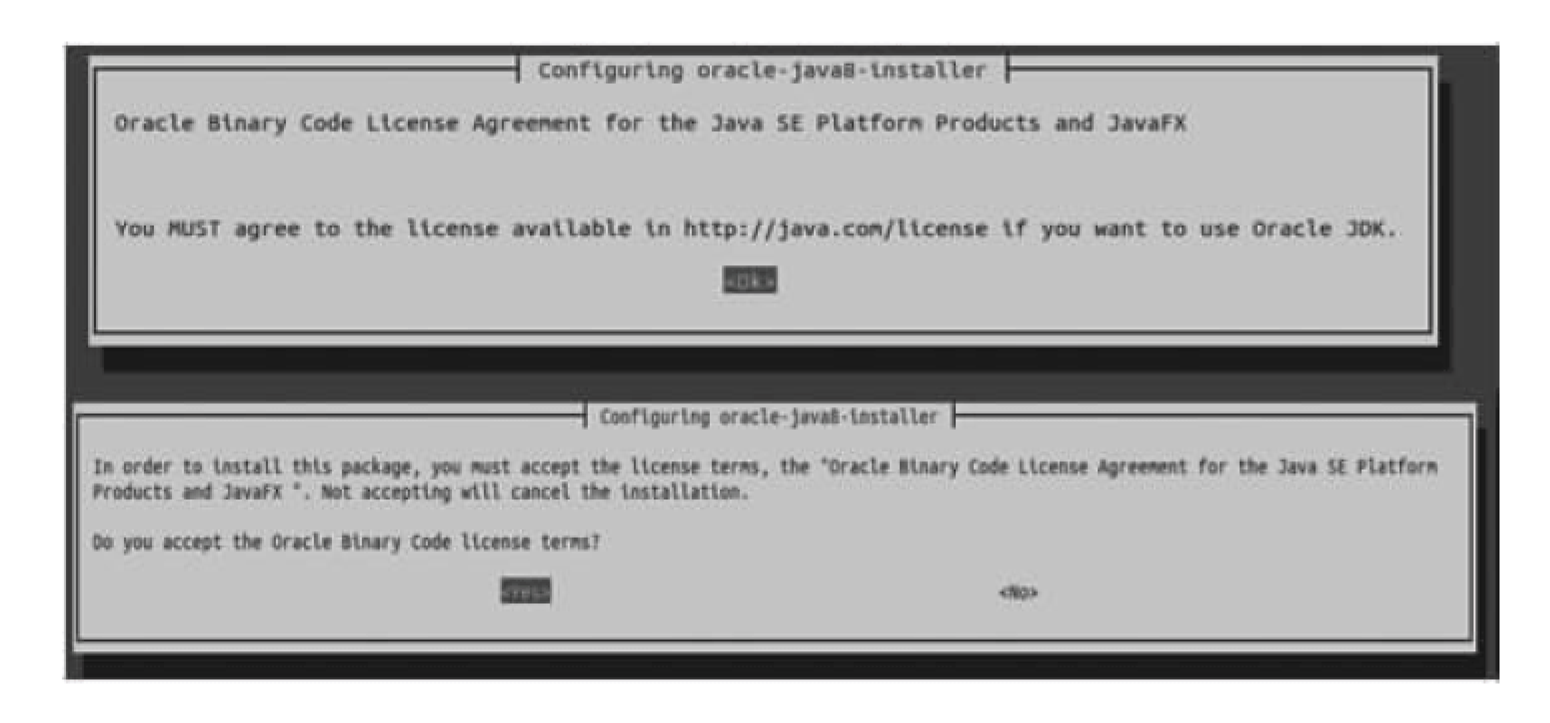
(2) 更新 APT 包，使之包括所有的最新文件。

```
sudo apt -get update
```

(3) 安装最新版本的 Oracle Java 8。

```
sudo apt -get -y install oracle-java8-installer
```

在安装过程中，可能要求接受 license 协议，如下图所示。



(4) 为了检验 Java 是否正确安装，可在终端输入下述命令。

```
java -version
```

---

① 译者注：APT(Advanced Packaging Tool)是 Linux 系统中的软件包管理程序，使用它可以找到想要的软件包，且安装、卸载、更新都很简便。

② 译者注：有些软件因为种种原因不能进入官方的 Ubuntu 软件仓库。为方便用户使用，允许用户通过 PPA 建立自己的软件仓库并自由上传软件。PPA 也被用来对一些打算进入 Ubuntu 官方仓库的软件或某些软件的新版本进行测试。

③ 译者注：sudo 是 Linux 系统管理指令，是允许系统管理员让普通用户执行 root 命令的工具。

```
yuvraj@ubuntu:~$ java -version
java version "1.8.0_74"
Java(TM) SE Runtime Environment (build 1.8.0_74-b02)
Java HotSpot(TM) 64-Bit Server VM (build 25.74-b02, mixed mode)
```

如上的截图表明 Java 已经成功安装。

1.5.1.2 在 Windows 上安装 Java

按如下步骤，在 Windows 环境下安装 Java。

(1) 使用如下链接，从 Sun Microsystems 中下载最新版 Java JDK。

```
http://www.oracle.com/technetwork/java/javase/downloads/index.html
```

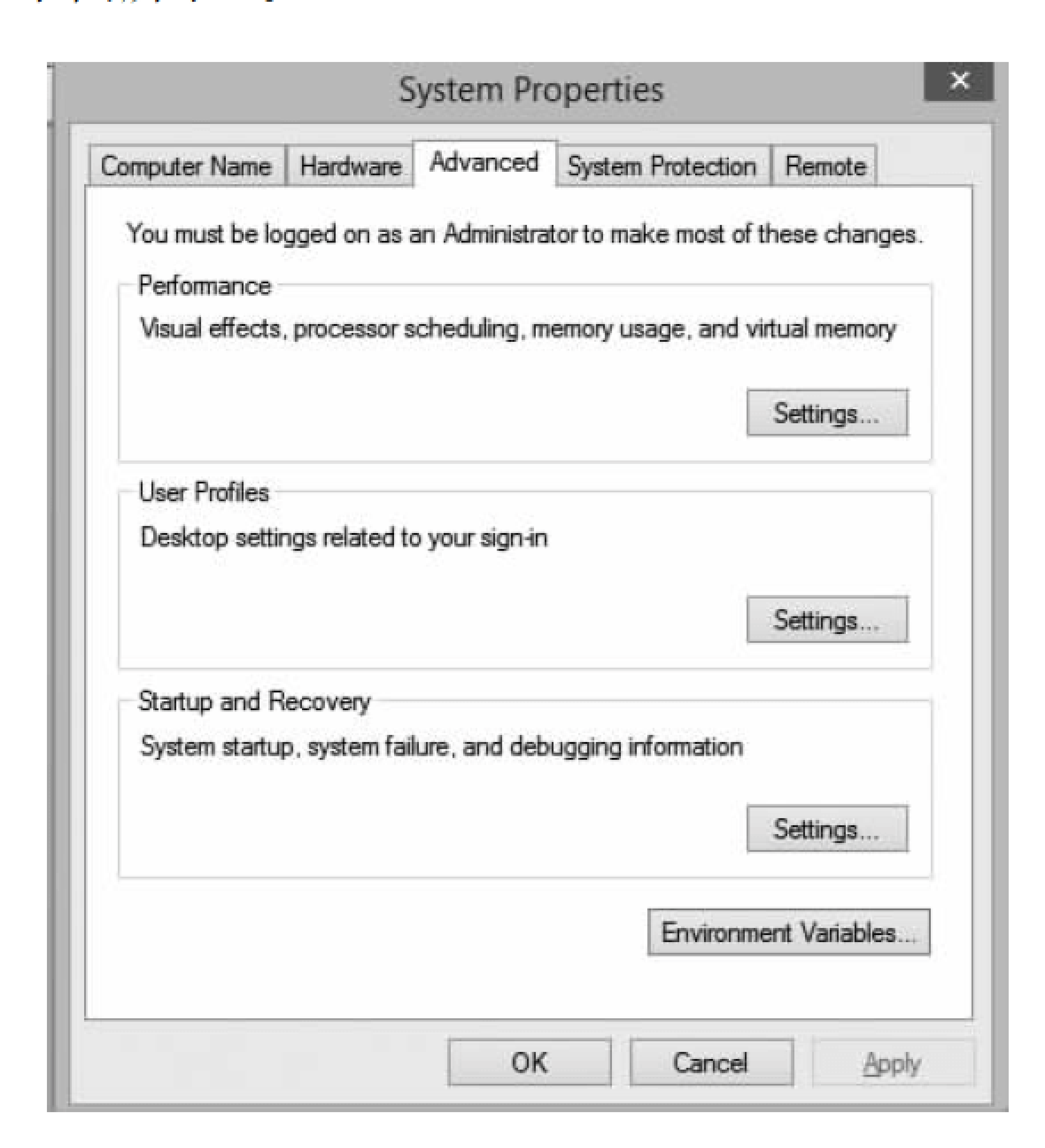
打开链接，单击 **Download** 按钮，下载 JDK。

可能被重定向到下载页：首先，单击 **Accept License Agreement** 无线按钮；之后，选择适合 Windows 系统的版本(32 位机用 x86，64 位机用 x64)，并下载对应的 EXE 文件。

(2) 双击安装文件，打开安装向导。

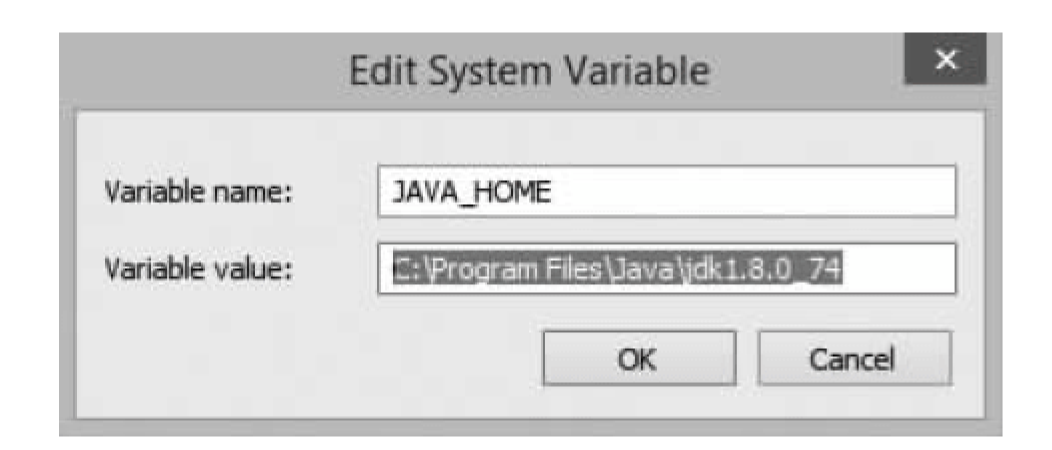
(3) 单击 Next 按钮，接受 license 协议；继续单击 Next 按钮，直到完成后续安装步骤。

(4) 为了能够运行 Java，还需要在 Windows 中设置 Java 环境变量。首先，打开"My Computer"的属性菜单，选择"**Advanced system settings**"。在"**Advanced**"选项卡上单击环境变量进行相应操作，如下图所示①。



① 译者注：原纸介质的图书上未有此图，但电子版上有。这里采用电子版的内容。

打开环境变量后，单击 **New**（在 **System Variables** 下面），为变量命名 JAVA_HOME，其变量值为 C:\ProgramFiles\Java\jdk1.8.0_74，如下图所示。检查一下自己的系统，确保 JDK 已经安装到指定的路径。



双击 **Path 变量**（在 **System Variables** 下），在文本框的最后，如果尚未指定 JDK 文件夹下 bin 文件夹的位置，请输入半角字符如下：%JAVA_HOME%\bin，并在后续打开的窗口中单击 **OK** 按钮。

> 请勿删除 path 变量文本框中已经存在的任何字符。

（5）为检验 Java 是否被正确安装，请在命令行提示符下输入如下内容：

```
java -version
```

```
C:\Users\ygupta>java -version
java version "1.8.0_74"
Java(TM) SE Runtime Environment (build 1.8.0_74-b02)
Java HotSpot(TM) 64-Bit Server VM (build 25.74-b02, mixed mode)
```

上图表明，Java 已经安装成功。

1.5.2　安装 Elasticsearch

本节将介绍 Elasticsearch v5.1.1 在 Ubuntu 和 Windows 环境下各自的安装方法。

1.5.2.1　在 Ubuntu 14.04 环境下安装 Elasticsearch

为了在 Ubuntu 下安装 Elasticsearch，请按如下步骤进行操作：

（1）以 debian package[①] 模式，下载 Elasticsearch 5.1.1：

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-
5.1.1.deb
```

① 译者注：广义的 debian 是指一个致力于创建自由操作系统的合作组织及其作品。这里的 debian 是 Linux 的一种发行版。和其他发行版相比，它拥有很多优势。

（2）输入如下命令，安装 debian package[①]。

```
sudo dpkg -i elasticsearch-5.1.1.deb
```

> Elasticsearch 将被安装在/usr/share/elasticsearch 文件夹下。配置文件位于/etc/elasticsearch。其 init 脚本位于/etc/init.d/elasticsearch。日志 log 位于/var/log/elasticsearch 目录。

（3）设置 Elasticsearch，使之能自启动。如果使用的是 System V 风格的，请运行如下命令：

```
sudo update-rc.d elasticsearch defaults 95 10
```

相应的提示将会出现在屏幕上：

```
Adding system startup for,    /etc/init.d/elacticsearch
```

使用下述命令，检查 Elasticsearch 的运行状态：

```
sudo service selasticsearch status
```

使用如下命令，以服务方式运行 Elasticsearch：

```
sudo service elasticsearch start
```

> 如果在前面安装了 ES-5.0.x 不支持的插件，Elasticsearch 将不会启动。由于插件不能使用，因此需要卸载早期 ES 版本中使用的插件。回到 ES 主目录，使用如下命令可移除插件[②]：bin/elasticsearch-plugin remove head。

Elasticsearch 命令的使用语法：

```
sudo service elasticsearch {start | stop | restart | force-reload | status}
```

如果正在使用的是 systemd 方式[③]，可按如下方式使用 Elasticsearch：

---

① 译者注：dpkg 是 Debian Packager 的简写，是为 debian 专门开发的套件管理系统，可方便软件的安装、更新及移除。所有源自 debian 的 Linux 发行版（例如 Ubuntu、Knoppix 等）都可使用 dpkg。

② 译者注：这里以 Head 插件为例介绍删除方法。Head 是一款在 ES 早期版本中使用的插件。ES 升至 5.0.x 后，对应版本的 Head 变为服务器端程序。该版本中官方推出 X-Pack 插件来提供与之相似的功能。使用该插件后，由于其中启用了 Security 组件，用户访问时需要提供身份验证信息，因此 X-Pack 被卸载之前，不具备身份验证功能的 Head 将无法连接 ES。

③ systemd 是用来启动守护进程 daemon（其中的 d 是 daemon 首字母）的工具。历史上，Linux 的启动一直采用 init 进程，但它启动时间长且脚本复杂。systemd 可部分解决这些问题，它取代了 init.d，成为系统的第一个进程（PID 为 1），其他进程都是它的子进程。

```
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable elasticsearch service
```

为了验证 Elasticsearch 是否已经正确安装，可在浏览器中打开 http://localhost:9200，或在命令行中运行如下命令：

```
curl -X GET http://localhost:9200
```

如果出现下图，则说明 Elasticsearch 安装成功。

```
{
  "name" : "xB20COp",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "wbXRP_h0QYK8QYNPJbalIA",
  "version" : {
    "number" : "5.1.1",
    "build_hash" : "5395e21",
    "build_date" : "2016-12-06T12:36:15.409Z",
    "build_snapshot" : false,
    "lucene_version" : "6.3.0"
  },
  "tagline" : "You Know, for Search"
}
```

1.5.2.2 在 Windows 环境下安装 Elasticsearch

请按照如下步骤，在 Windows 环境下安装 Elasticsearch。

(1) 使用如下链接，下载 Elasticsearch 5.1.1。

```
https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-5.1.1.zip
```

打开链接，下载 ZIP 文件包。

(2) 解压下载的 ZIP 文件包。可以使用 WinRAR、7-Zip 以及其他解压软件完成解压工作（如果没有上述软件，请自行下载）。

解压后，会在对应目录中展开相应的文件和文件夹。

(3) 打开解压的文件夹，导航到其中的 bin 文件夹。

(4) 单击其中的 elasticsearch.bat 文件，运行 Elasticsearch。

> ℹ 如果关闭此运行窗口，Elasticsearch 的运行节点将会被停止，Elasticsearch 也将停止运行。

(5) 为验证 Elasticsearch 是否正确安装，请在浏览器中输入：http://localhost:9200。如果出现下图，说明安装成功①。

① 译者注：原纸介质的图书上未有此图，但电子版上有。这里采用电子版的内容。

```
{
  "name" : "3Fqp-HA",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "V0u-BkqBTLerMhToip7rGQ",
  "version" : {
    "number" : "5.1.1",
    "build_hash" : "5395e21",
    "build_date" : "2016-12-06T12:36:15.409Z",
    "build_snapshot" : false,
    "lucene_version" : "6.3.0"
  },
  "tagline" : "You Know, for Search"
}
```

按如上所述方法安装 Elasticsearch 后，在命令行方式下，进入其 bin 文件夹，使用如下命令，以服务（service）的方式启动 Elasticsearch：

```
elasticsearch-service.bat install
Usage: elasticsearch-service.bat install | remove | start | stop | manager
```

1.5.3 安装 Kibana

本节将分别介绍在 Ubuntu 和 Windows 环境下安装 Kibana 5.1.1 的方法。在安装之前，需要如下预备环节：

- Elasticsearch 已经安装正确，且运行在 9200 端口（默认配置的端口）。
- 确保 Kibana 运行的端口——默认是 5601 端口——未被其他应用所占用。

1.5.3.1 在 Ubuntu 14.04 环境下安装 Kibana

请按照如下步骤，在 Ubuntu 环境中安装 Kibana。

（1）在安装 Kibana 之前，请确认计算机系统是 32 位还是 64 位。可以通过如下命令得到相关信息：

```
uname -m
```

如果输出 x86_64，表示计算机是 64 位的系统；如果输出 i686，表明计算机是 32 位的系统。

（2）以 debian package 方式下载 Kibana 5.1.1：

- 对于 64 位系统：

```
wget https://artifacts.elastic.co/downloads/kibana/kibana-5.1.1-amd64.deb
```

- 对于 32 位系统：

```
wget https://artifacts.elastic.co/downloads/kibana/kibana-5.1.1-i386.deb
```

（3）使用如下命令安装 debian package：

- 对于 64 位系统：

```
sudo dpkg -i kibana-5.1.1-amd64.deb
```

- 对于 32 位系统：

```
sudo dpkg -i kibana-5.1.1-i386.deb
```

> Kibana 将会被安装在/usr/share/kibana 文件夹下。配置文件位于 etc/kibana。其 init 脚本位于 etc/init.d/kibana。日志文件位于/var/log/kibana 文件夹。

(4) 配置 Kibana 使之自启动。如果使用 SysV init 分布式处理方式，可输入如下命令：

```
sudo update-rc.d kibana defaults 95 10
```

上述命令将会在屏幕输出：

```
Adding system startup for /etc/init.d/kibana
```

使用下述命令，检验 Kibana 的状态：

```
sudo service kibana status
```

使用下述命令，以服务(service)方式运行 Kibana：

```
sudo service kibana start
```

Kibana 命令的使用方式如下：

```
sudo service kibana {start|force-start|stop|force-stop|status|restart}
```

如果正在以 systemd 分布式方式运行，可使用如下命令：

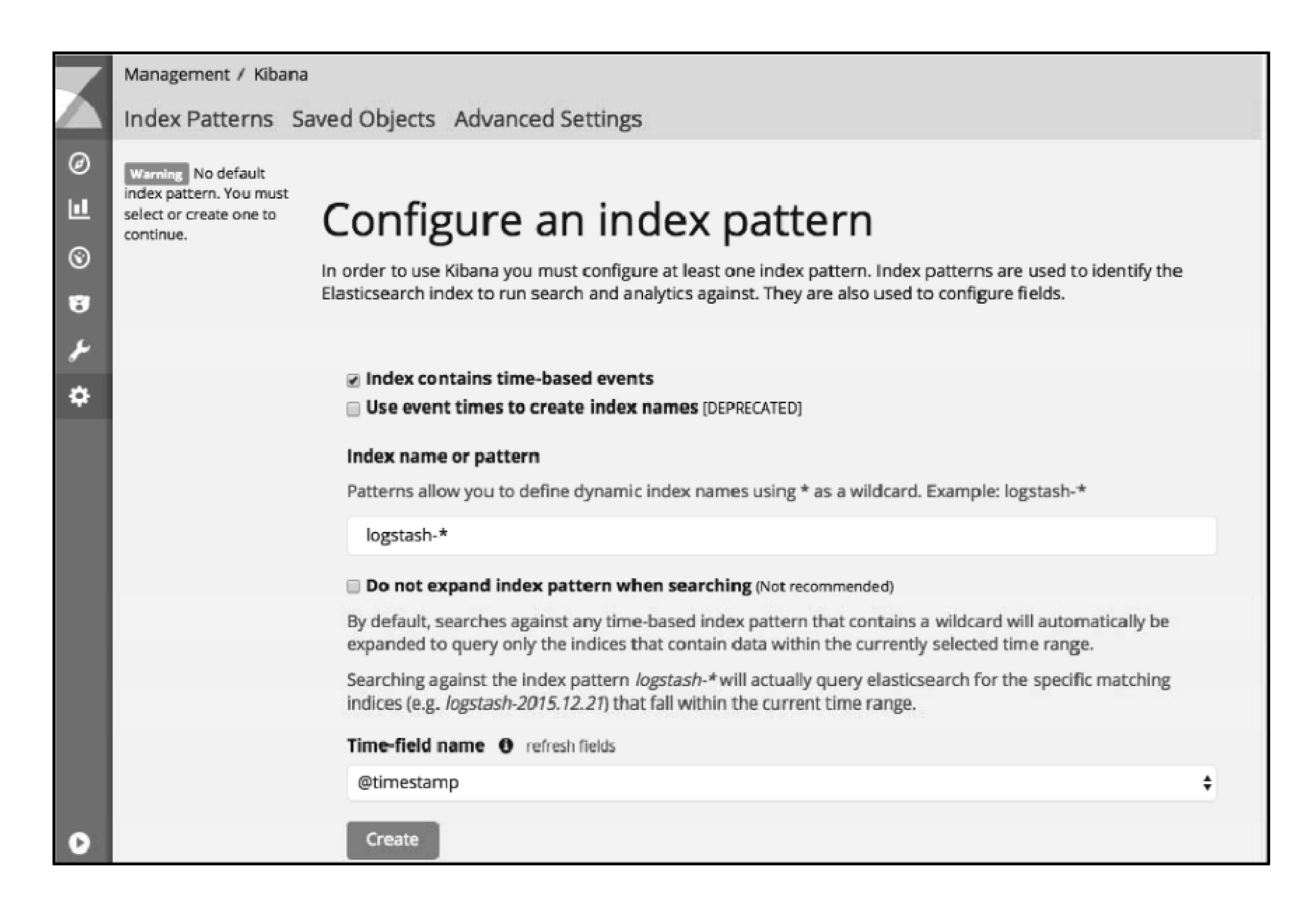
```
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable kibana.service
```

> TIP 如果想安装其他任何版本的 Kibana，可访问 Elastic 团队提供的软件下载网站，复制相应的 debian package 链接，使用 wget 来获取安装包。

(5) 可以在浏览器中输入 http://localhost:5601，打开如下图所示的链接地址，验证 Kibana 是否正确安装。

1.5.3.2　在 Windows 环境下安装 Kibana

请按照如下步骤，在 Windows 环境下完成 Kibana 的安装。



（1）使用如下链接，从 Elastic 网站下载 Kibana 5.1.1。

https://artifacts.elastic.co/downloads/kibana/kibana-5.1.1-windows-x86.zip

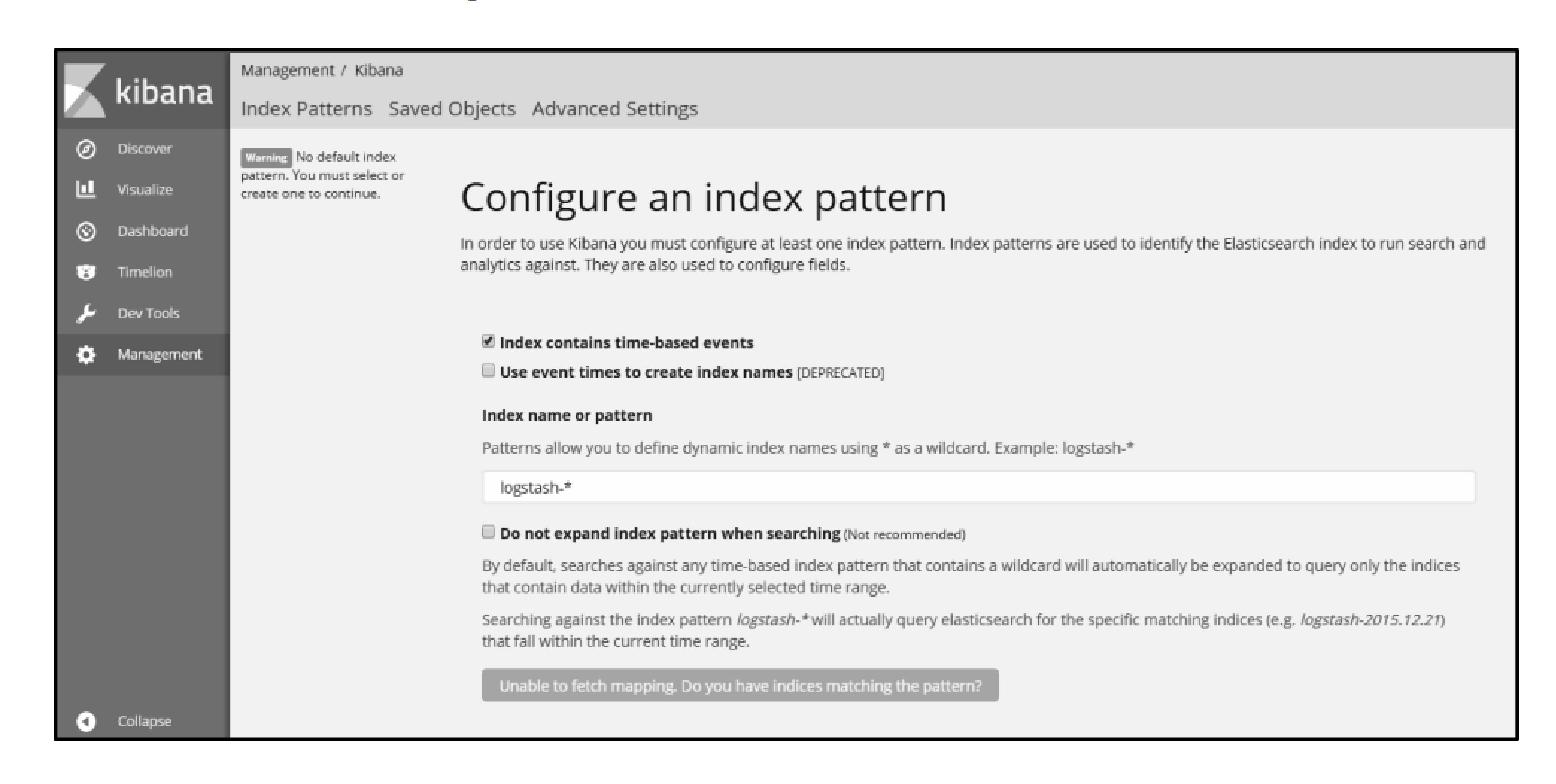
打开链接，单击下载相应的 ZIP 文件包。

（2）使用 WinRAR、7-ZIP 或类似的相关软件解压 ZIP 文件。之后，会在相应的文件夹下解压出 Kibana 对应的文件和文件夹。

（3）在解压出的文件夹中，定位到其中的 bin 文件夹下。

（4）单击 kibana.bat 文件，运行 Kibana。

（5）在浏览器中输入 http://localhost：5601，验证 Kibana 是否正确安装，如下图所示。



1.5.4　安装 Logstash

本节分别介绍在 Ubuntu 和 Windows 环境下安装 Logstash 5.1.1 的方法。

1.5.4.1　在 Ubuntu 14.04 下安装 Logstash

请按照如下步骤，完成在 Ubuntu 环境下 Logstash 的安装。

(1) 以 debian 方式下载 Logstash 5.1.1：

```
wget https://artifacts.elastic.co/downloads/logstash/logstash-5.1.1.deb
```

(2) 使用如下命令安装 debian package。

```
sudo dpkg -i logstash-5.1.1.deb
```

> Logstash 将会被安装到 usr/share/logstash 文件夹下。其配置文件位于 etc/logstash。日志 Log 文件位于/var/log/logstash 目录下。

(3) 使用如下命令检查 Logstash 的状态。

```
sudo initctl status Logstash
```

使用如下命令，以服务(service)方式运行 Logstash。

```
sudo initctl start logstash
```

> Logstash 将会被安装到 usr/share/logstash 文件夹下。

1.5.4.2　在 Windows 下安装 Logstash

按如下步骤，在 Windows 环境下安装 Logstash。

(1) 使用如下链接从 Elastic 网站下载 Logstash 5.1.1。打开链接，下载 ZIP 压缩文件包。

```
https://artifacts.elastic.co/downloads/logstash/logstash-5.1.1.zip
```

(2) 使用 WinRAR、7-ZIP 或类似的解压软件解压下载的 ZIP 文件包。这将会在相应的目录中解压出 Logstash 文件和文件夹。

(3) 打开解压后的文件夹，导航到其中的 bin 子文件夹。

(4) 为验证是否正确安装 Logstash，可定位到 bin 文件夹，在命令行提示符后输入：

```
logstash -version
```

如果正确安装了 Logstash，将会显示 Logstash 的版本等信息。

1.5.5 安装 Filebeat

本节分别介绍在 Ubuntu 和 Windows 环境下安装 Filebeat 5.1.1 的方法。

1.5.5.1 在 Ubuntu 14.04 下安装 Filebeat

请按照如下步骤，完成在 Ubuntu 环境下的 Filebeat 的安装。

（1）在安装 Filebeat 之前，请确认计算机系统是 32 位还是 64 位，命令如下：

```
uname -m
```

如果输出 x86_64，表示计算机是 64 位的系统；如果输出 i686，表明计算机是 32 位的系统。

（2）以 debian 方式下载 Filebeat 5.1.1。

- 对于 64 位系统：

```
wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-5.1.1-amd64.deb
```

- 对于 32 位系统：

```
wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-5.1.1-i386.deb
```

（3）使用如下命令安装 debian 模式的软件包：

- 对于 64 位系统：

```
sudo dpkg -i filebeat-5.1.1-amd64.deb
```

- 对于 32 位系统：

```
sudo dpkg -i filebeat-5.1.1-i386.deb
```

> Filebeat 将会被安装在/usr/share/filebeat 文件夹下。配置文件位于 etc/filebeat。其 init 脚本位于 etc/init.d/filebeat 文件夹。日志文件位于/var/log/filebeat 文件夹。

（4）配置 Filebeat，使之能自启动。如果使用 SysV init 分布式处理，可输入如下命令：

```
sudo update-rc.d filebeat defaults 95 10
```

上述命令运行后，将会在屏幕输出：

```
Adding system startup for /etc/init.d/filebeat
```

使用如下命令，检验 Filebeat 的状态：

```
sudo service filebeat status
```

使用下述命令，以服务(service)方式运行 Filebeat：

```
sudo service filebeat start
```

Filebeat 对应的命令使用方式如下：

```
sudo service filebeat {start|stop|status|restart|force-reload}
```

> 如果想以服务(service)方式运行 Filebeat，需运行配置文件/etc/filebeat/filebeat.yml。

> 如果想安装其他任何版本的 Filebeat，可访问 Elastic 团队提供的软件下载网站，复制 debian 软件包链接，使用 wget 来获取安装包。

1.5.5.2　在 Windows 环境下安装 Filebeat

按照如下步骤，在 Windows 环境下安装 Filebeat。

(1) 安装 Filebeat 之前，请在命令行提示符后输入如下命令，检查系统是 32 位的还是 64 位的。

```
wmic os get osarchitecture
```

系统将会输出是 64 位或 32 位的提示信息。

(2) 使用如下链接从 Elastic 网站下载 Filebeat 5.1.1。

对于 64 位系统：

```
https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-5.1.1-windows
-x86_64.zip
```

对于 32 位系统：

```
https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-5.1.1-windows
-x86.zip
```

单击相应的链接，下载相应的 ZIP 文件。

(3) 使用 WinRAR、7-ZIP 或类似的解压工具软件解压下载的 ZIP 文件包，这将会在相

应的目录中解压 Filebeat 压缩包中的文件和文件夹。

(4) 打开 Windows 的 PowerShell 用作超级用户管理员(如果没有,请安装)[①]。

(5) 定位到 Filebeat 的文件夹(例如 C:\Users\username\Desktop)。在 Windows 的 PowerShell 中运行如下命令:

```
.\install-service-filebeat.ps1
```

> 如果解析的脚本无法在系统上执行,那么需要在当前 Session 上完成设置,以便脚本能正常执行。例如:PowerShell.exe -ExecutionPolicy UnRestricted-File.\install-service-filebeat.ps1。

这将会安装 Filebeat,并使之成为 Windows 的服务(service)。

1.6 X Pack 简介

还有一些和 Elastic Stack 相关的特性也需要重视,其中应特别关注安全(Security)、监控(Monitoring)、报警(Alerts)等。**X-Pack** 包括如下特性:

- 安全(Security);
- 报警(Alerts);
- 监控(Monitoring);
- 可视化(Graphs);
- 报告(Reporting)。

安全(Security)、报警(Alerts)以及监控(Monitoring)是之前就出现过的组件,只是在早期它们的名称分别为 Shield、Watcher、Marvel,现在又加入了可视化(Graphs)、报告(Reporting)等。作为一个团队,它们现在统称为 X-Pack。像 Elastic Stack 中的工具一样,它们也能以相同的版本执行开发、构建、测试、发布等功能。

> 本书中所有使用的代码可以在 Github 软件库中下载[②]:
>
> https://github.com/kravigupta/mastering-elastic-stack-code-files/tree/5.1.1

① 译者注:Windows PowerShell 是一种命令行外壳程序和脚本环境,扩展了在 Windows 命令提示符和 Windows Script Host 环境中的功能。

② 译者注:原文电子版中提供了此链接。

1.7　本章小结

本章介绍了 Elastic Stack 及其组件,包括 Elastic Stack 是如何发展的,它改变了什么,它又推出了什么,以及 ELK Stack 如何演变为 Elastic Stack。我们也了解了这些组件是如何帮助相关行业组织满足业务需求的一些案例。

在本章的后面部分,介绍了如何设置和配置 Elasticsearch、Logstash 和 Kibana,以及作为服务的 Filebeat。最后,本章简单介绍了 X-Pack,其详细内容在本书后续部分将会介绍。

下一章中,我们将会学习关于 Elasticsearch 的一些更详细的内容,包括 API、QueryDSL 等。



走进 Elasticsearch

在前面的章节，我们了解了 Elasticsearch、Logstash、Kibana 和 Beats 的基本内容，以及如何安装和设置这些组件以建立数据管道。我们也逐渐了解了 Elasticsearch 的作用及其与 Stack 中的其他组件一起工作的方式，不过那只是冰山一角。为了更好地理解 Elasticsearch 是如何工作的，我们需要了解它的 API、模块和插件。这些内容分为两章介绍。

在本章，我们将深入到 Elasticsearch 中，内容涉及以下话题：

- Elasticsearch 的起源；
- 理解体系结构；
- Elasticsearch API 的用法；
- 聚合；
- Painless 脚本说明。

在本章最后，将介绍如何使用聚合 aggregations，并体会 API 强大的性能。更多关于 Elasticsearch 的信息，将在第 8 章 Elasticsearch API 中介绍。

2.1 Elasticsearch 的起源

这一切都源于 Lucene，它是一个由 Apache 软件基金会支持的优秀项目。目前许多项目都是基于 Lucene 的，例如 Apache Solr、Elasticsearch、Apache Nutch、Lucene. Net、DocFetcher 等。如果试图找到一种搜索引擎的解决方案，肯定会对 Lucene 印象深刻。它不仅可用于 Java 环境中，还可以用于 Delphi、Perl、C＃、C++、Python、Ruby 以及 PHP 环境中。Lucene 的完整实现详见 http://wiki.apache.org/lucene-Java/LuceneImplementations。

Lucene 是全文搜索引擎，它可以在文档中创建索引。在一段或海量的文本中，每一个字符串称为一个**词项**（**term**），一个词项序列称为一个**字段**（**field**），而一个字段序列称为一个**文档**（**document**）。一个**索引**（**index**）包含一系列文档，数据索引也作为文档。

在书中，通常看到一个索引将所有的关键字列出，这有助于我们找到实际的内容。这种

类型的索引称为**倒排索引**(**inverted-index**)，其中使用词项或字符串来索引文档。

Lucene是一个很不错的基于文本的搜索引擎项目。它最早出现在1999年，从那时起，有大量项目基于Lucene而实现。值得注意的一点是，有些搜索引擎将Lucene作为其核心。这些项目通过包装(wrapping)来扩展Lucene，为它创建一个接口，添加更多的特性等，从而提供各种各样的解决方案。

对于Java的基础项目，Apache Solr和Elasticsearch是个不错的选择。可以在网上找到很多讨论搜索引擎优越性的信息。

在Elasticsearch之前，Shay Banon创建的**Compass**也是建立在Lucene上的搜索引擎项目。Compass拥有十分卓越的特性，例如无缝整合、XML、JSON支持，以及整合ORM库(如Hibernate和JPA)的功能，这些特性可以使Java开发者很容易对其进行开发。当Compass升级到3.0版本时，Shay认为它需要做一些重大改变，应将其所基于的Lucene升级至2.9版本，以解决可扩展性的问题。他想到了一个更好的解决方案，这个方案能够解决所有的问题，因此Elasticsearch出现了，并代替了Compass 3.0。2010年7月，他在博客写了一篇名为《Compass和Elasticsearch的未来》的文章(http://thedudeabides.com/articles/the_future_of_compass)。在文章中他写道："因此，我开始构建Elasticsearch，它基本上是一个完完全全的分布式解决方案。我还想创建一个可以被其他编程语言轻松使用的搜索解决方案，这就意味着在HTTP之上表示JSON，这种方法不用牺牲它在Java编程语言中的易用性(或者更特殊的情况，例如JVM)。"

Elasticsearch一出现，就开始受到开源社区中众多开发者的关注。因此，相关领域中涌现了大量使用Elasticsearch的客户端。其中GitHub、Quora、Stack Exchange、Mozilla、StumbleUpon、CISCO和Netflix是最著名的。更全面的名单可以访问以下产品网站查看：https://www.elastic.co/use-cases。

Elasticsearch被认为是最先进的搜索引擎，它能提供Lucene提供的任何内容，而且远不止这些。让我们来看看其中的关键特性：

- **JSON支持**：Elasticsearch采用JSON格式的文档作为输入来创建索引。默认情况下，所有字段的属性都会被自动检测和索引。Elasticsearch可以为存储的数据创建映像(默认设置为字符串，可以人为更改)。它不需要定义模式(Solr中使用schema.xml来配置)。由于Elasticsearch利用了Lucene的优点，因此可以提供对已索引的数据执行全文检索的功能。
- **RESTful API**：在使用JSON数据时，RESTful API可以执行大部分必要的操作。可以通过发送JSON文档来向索引添加数据、删除条目、更新条目，以及执行其他更多的操作。关于API，我们将在本书中的后续章节中进行讨论。
- **实时数据可用性和分析**：一旦数据被索引，它就可以用于搜索和分析，这些都是实

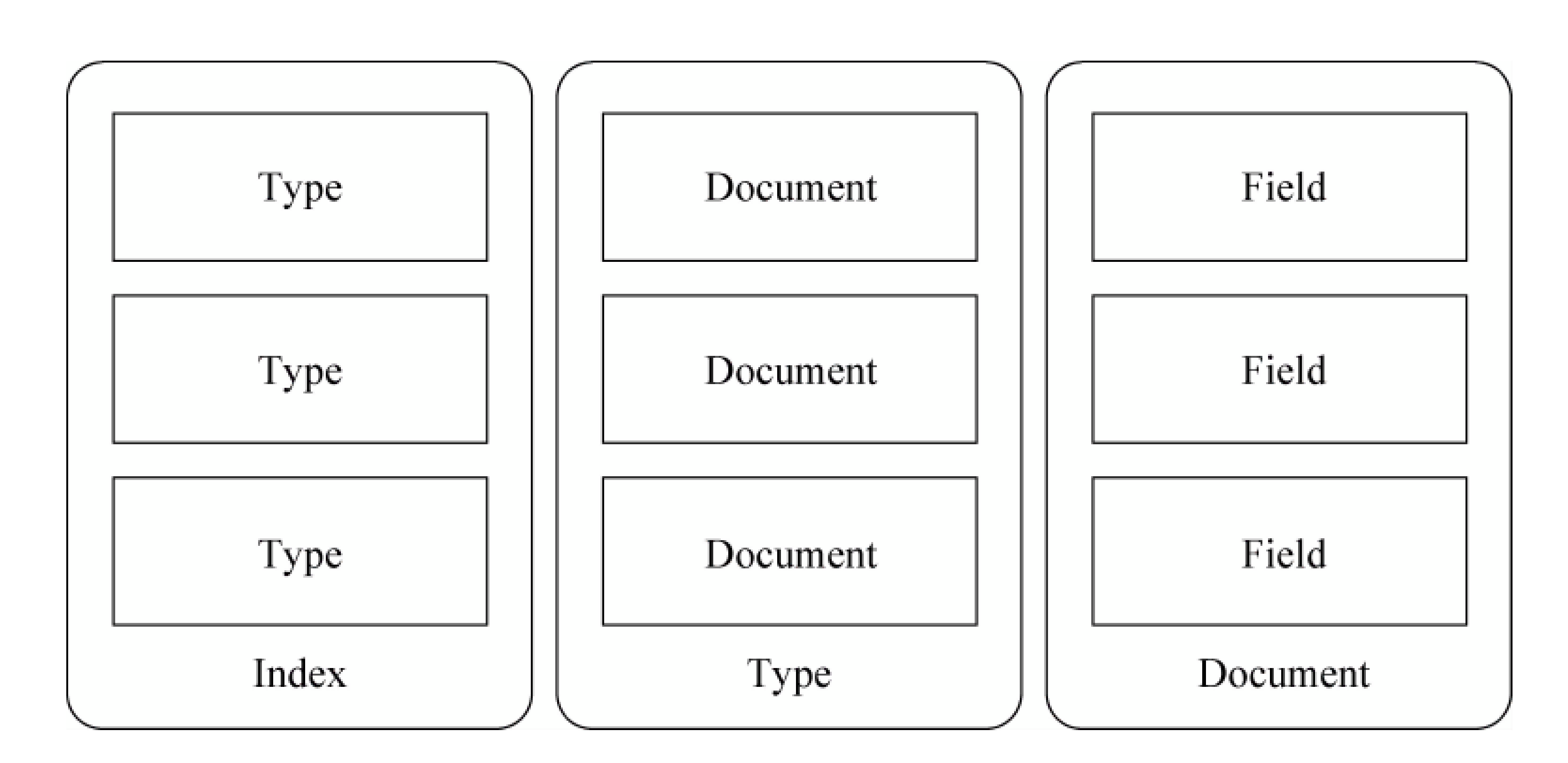
时的。

- **分布式**：Elasticsearch 允许设置尽可能多的节点来满足我们的需求。集群将管理一切，并且它可以横向增长到一个很大的数量（如 1000 节点）。要扩展集群，只需在具有相同集群名称的网络中启动另一个节点，然后它将会添加到该集群中。
- **高可用性**：集群足够智能，可以检测从集群中添加/删除的新节点或失效节点。当一个节点被添加或删除后，程序能以某种方式重新均衡这些数据，使整个集群仍然可用。
- **数据安全第一**：数据的任何变化都会记录在事务日志中，不仅在单个节点上，而且在多个节点上（在有节点发生故障的情况下，它仍然可用）。以这种方式，Elasticsearch 试图把数据丢失的概率降到最小。
- **多租户技术**：在 Elasticsearch 中，可以为索引创建别名。通常一个集群包含多个索引，这些别名允许筛选一个索引来实现多租户技术。

2.2 了解 Elasticsearch 的体系结构

为了了解 Elasticsearch 如何工作，我们必须明白它的体系结构。

为了理解索引（Index）、类型（Type）、文档（Document）和字段（Field）如何协同工作，让我们参考下图：



如上图所示，**索引**包含一个或多个类型。**类型**可以认为是关系型数据库中的一个表。类型有一个或多个**文档**，文档中有一个或多个字段，字段是由键值对构成的。

集群有一个或多个节点。集群由它们的名字来标识。默认情况下，集群的名字是 elasticsearch。如果必须在同一网络中设置多个 Elasticsearch 实例，则应该为不同的集群设置不同的名称，否则所有节点将加入同一个集群。与集群类似，节点也有名称。可以给它分配一个名称和一个集群名。如果不提供群集名称，那么节点将自动搜索并加入名为

elasticsearch 的集群中。

如果不为节点提供名称，那么程序会分配给该节点一个 ID 号。这是一个随机的**通用唯一识别码(UUID)**，并且节点将选择这个 ID 号的前 7 位作为其名称，这样每个节点都会是唯一的。

有一种情况可能会发生：一个索引中可能存储大量数据，这样的数据量可能超过一个集群节点所在硬件的容量。对于这种情况，索引可以分成多个分片(shard)。

分片的类型有两种：主分片和副本分片。每一个文档在存入索引时，首先被添加到主分片，然后添加到一个或多个副本分片中。如果为集群设置了多个节点，那么副本分片将位于不同的节点上。

默认情况下，Elasticsearch 会创建五个主分片，每个主分片有一个副分片。因此，对于一个索引，如果没有指定，那么总共将创建 10 个分片。可以更改这个配置，该内容将在本章后面的 Indices API 部分中讲到。

2.2.1 推荐的集群配置

要搭建一个能够正常运行的集群，应该考虑一些配置问题。正如前面已经讨论过的，应该为集群设置一个集群名：

```
cluster.name: my-cluster-name
```

同样，每个节点应该被赋予唯一的名称，以便识别：

```
node.name: node-1
```

另外还有其他与数据路径、节点发现等相关的设置。

2.2.1.1 最小主节点

在所有节点被称为集群之前，可以指定一个属性，让 Elasticsearch 去找出具有主节点资格的最小节点数目。虽然 Elasticsearch 对这些属性有默认设置，但是计划搭建一个 Elasticsearch 集群时，应该考虑这个设置。

使用此设置是为了防止数据丢失。在网络故障或其他情况下，集群可能选择另一个主节点，而第一个主节点仍处于运行状态，但无法访问到。这就产生了一种被称为脑裂(split brain)的故障，集群被分成两个不同的集群并导致数据丢失。

为了避免脑裂问题发生，应在集群中使用此设置。

设定的值可以由此公式得出：(总节点数/2)+1。

如果打算用 4 个节点建立一个集群，就应该将这个值设置为 3：

```
discovery.zen.minimum_master_nodes: 3
```

所有这些节点应将 node. master 设置为 true：

```
node.master: true
```

在第 8 章 Elasticsearch API 中的模块部分，将学习更多关于节点类型的内容。

2.2.1.2　本地集群设置

有时会尝试在同一服务器上设置多个实例，这些实例可以共享包含索引、分片、集群元数据等的同一个数据目录。为了在实例之间共享数据，每个节点都可以配置以下代码：

```
path.data: /path/to/data/directory
```

通常，当 Elasticsearch 从 ZIP/RAR 包中解压并使用时，程序将使用其存储目录中的 data 文件夹作为数据目录。最佳的做法是，设置数据目录位于 Elasticsearch 的安装目录之外。

如果希望将一个节点设置为数据节点，只需进行如下配置：

```
node.data: true
```

然后将 node. master 设置为 false。

默认情况下，Elasticsearch 不允许多个节点共享同一个数据目录。如果要共享相同的数据，请使用以下设置：

```
node.max_local_storage_nodes:2
```

上述设置代码允许两个实例共享数据。然而，应该记住不同类型的节点，如主节点、数据节点等，不应该共享相同的数据目录。

> ℹ 不推荐在一台机器上设置多个 Elasticsearch 实例的集群配置。

2.2.2　了解文档处理

正如我们所知道的，无论何时创建索引，都会创建主分片和副本分片，每个分片都可以有多个副本。这个完整的组也称为副本组（replication group）。在副本组中，主分片可以作为任何对文件执行索引操作的切入点，它能确保文档数据和相关的操作是有效的。如果一切正常，操作将会执行，主分片上的操作会重复执行在副本分片上，这些职责都是由主分片完成的。

一个文档没有必要被复制到所有副本分片上。相反，Elasticsearch 会保留应接受操作的副本分片。这个分片的列表也被称为同步副本（in-sync copies），它由主节点维护。通过维护这个列表，可以确保操作是由这些分片执行的，并且用户也得到了确认信息。这个处理

模型也称为**数据复制模型**(**data-replication model**),它是基于**主备份模型**(**primary-backup model**)的。

在主分片发生故障的情况下,该分片所在的节点会向主节点发送一个消息,将错误信息告知它。在此期间,索引操作不会执行,所有分片会等待主节点在副本分片中定义一个主分片出来。

除此之外,主节点也会检查所有分片的健康状况,并且可以将(可能由于网络故障或断开导致的)健康状况不佳的主分片降级。主节点会通知另一个节点开始建立一个新的分片,以便于使系统恢复到健康的状态,即绿色状态。

2.3　Elasticsearch API

很多 API 都可用于管理 Elasticsearch,这些 API 帮助我们管理集群、索引、搜索等。在本节中,我们将详细地了解每一个 API。

我们可以通过命令提示符、Kibana 的控制台,或者任何可以调用 RESTful API 的工具来使用这些 API。

> 默认情况下,Elasticsearch 在 9200 端口监听 HTTP 请求,Kibana 使用相同的端口来连接到 Elasticsearch。要了解更多关于控制台的内容,请参阅第 4 章 Kibana 界面中的探索开发者工具 Dev Tools 部分。

Sense 是 Kibana 中的一个强大的插件,它允许使用网络接口来调用 Elasticsearch API。我们将在第 8 章 Elasticsearch API 学习 Sense 插件的内容。在本章中,将使用命令提示符程序 **curl** 来通过访问 HTTP 请求访问这些 API。

一个典型的 curl 请求包含一个动词(即请求方式)、URL 和消息体:

```
$ curl -X{Verb} 'url' -d '{message-body}'
```

动词 Verb 可以是 GET、PUT、POST、DELETE 和 HEAD,URL 可以是 HTTP(默认)或者 HTTPS。消息体通常是要索引的文档、执行的查询,或者是要让 Elasticsearch 执行的命令。

2.3.1　有关文档的 API

这些 API 允许用户向索引中添加文档,获取已有文档,以及编辑和删除操作。这些 API 分为两组:单文档 API 和多文档 API。

2.3.1.1　单文档 API

在一个文档上执行操作时,这些 API 是适用的。它们可进一步划分为:

- 索引文档 Index API；
- 获取文档 Get API；
- 删除文档 Delete API；
- 更新文档 Update API。

1. 索引文档 Index API

Index API 可以将一个 JSON 文档添加到索引中。为了理解这一点，用图书馆来举一个例子。向图书馆中添加一本图书：

```
$ curl -XPUT 'http://localhost:9200/library/book/1?pretty' -d '{
        "author" : "Ravi Kumar Gupta",
        "title" : "Test-Driven Javascript Development",
        "pages" : 240
}'
```

> 命令中 URI 的结尾使用了 pretty 参数，来输出缩进格式良好的 JSON 结果（如果有的话）。

如果索引不存在，程序将自动创建一个名为 library 的索引，并将一本书的信息添加到这个索引中。如果 library 索引已经存在，那么它会将文档添加进去，除非其中已经有一个相同 id 的文档存在。

请看下面的这段代码：

```
$ curl -XPUT http://localhost:9200/library/book/1?op_type=create&pretty' -d '{
        "author" : "Ravi Kumar Gupta",
        "title" : "Test-Driven Javascript Development",
        "pages" : 240
      }'
$ curl -XPUT 'http://localhost:9200/library/book/1/_create?pretty' -d
  '{
        "author" : "Ravi Kumar Gupta",
        "title" : "Test-Driven JavaScript Development",
        "pages" : 240
      }'
```

在上面的代码的第一个命令中，指定了 op_type 参数来设置操作的类型，并将其值设置为 create，也可以通过在 URI 的结尾添加/_create 来实现执行的操作。两者会有相同的输出，并且此文档会被添加到索引中。默认情况下，如果没有任何具体说明（如前面代码中的第一个操作），这个操作将设置为 create。如果索引原本不存在，那么前面提到的

三个命令中的任何一个都能够创建一个 library 索引，并添加一个包含作者、标题和页数字段的条目。

> 使用_create 或 op_type 参数来显式地使用创建命令时，如果索引中已经存在具有相同的 id 的文档，程序就会抛出 version_conflict_engine_exception 异常。

让我们分析一下前面代码的输出：

```
{
    "_index" : "library",
    "_type" : "book",
    "_id" : "1",
    "_version" : 1,
    "result" : "created",
    "_shards" : {
        "total" : 2,
        "successful" : 1,
        "failed" : 0
    },
     "created" : true
}
```

这样的输出表明文档已添加到名为 library 的索引中，并且指定的类型(type)为 book。文档的 id 为 1，version 为 1，result 的值设置为 created，created 值为 true 表明操作成功。如果索引中相同的文档已经存在，执行同样的操作时，将得到 result 的值为 updated，_version 将更新到下一个值，在这里就是 2，并且 created 的值将为 false。这里的 shard 显示其总共有两个分片，其中文件被成功复制到一个分片上。如果成功结果的值至少为一个，那么操作就是成功的。

在默认情况下会创建五个分片，每个分片有一个副本。在五个分片中，有两个被选择用于索引文档。执行成功的分片数量也取决于集群设置。如果只有一个节点，那么副本将不会出现。另外要注意到，此时集群/节点的健康状态是黄色的，即亚健康状态，那么成功的值往往小于分片数。如果在集群中设置了多个节点，则值可能是相同的。下面是在两个节点的集群处理相同命令的输出结果：

```
{
    "_index": "library",
    "_type": "book",
    "_id": "1",
    "_version": 1,
```

```
    "result": "created",
    "_shards": {
        "total": 2,
        "successful": 2,
        "failed": 0
    },
    "created": true
}
```

正如我们所看到的，文档被正确地复制了。需要的是输出信息中 successful 字段后面的一个正值。只要有一个分片/副本可用，文档就可以被复制到多个分片/副本中。

请注意，已经使用一条 URI，即/library/book/1/_create，添加了一个值为 1 的 id。如果想自动生成这个 id，我们可以在 curl 中使用 POST 方式来代替 PUT 方式。

如果使用 POST 方式执行，那么即使索引不存在，它也会自动创建索引和 id。

让我们来添加另一本书的信息：

```
$ curl -XPOST 'http://localhost:9200/library/book?pretty' -d '{
    "author" : "Yuvraj Gupta",
    "title" : "Kibana Essentials",
    "pages" : 210
}'
```

现在，分析一下输出信息会是什么内容：

```
{
    "_index" : "library",
    "_type" : "book",
    "_id" : "AVSPSXXDxiIiqkaLJfTy",
    "_version" : 1,
    "result" : "created",
    "_shards" : {
        "total" : 2,
        "successful" : 1,
        "failed" : 0
    },
    "created" : true
}
```

一切都是相似的，但 id 的值是由 Elasticsearch 自动生成的。

默认情况下，Elasticsearch 基于文档的 id 决定将文档放在哪一个分片上。使用文档的 id，程序会计算出一个哈希值，并决定选择一个还是多个分片。为了替代 Elasticsearch 的基

于 id 做出的决定，可以提供一个值来用于计算哈希值。分片分配 shard allocation 的过程称为 routing，我们可以在查询的 URI 中使用一个 routing 参数，如下面的命令所示：

```
$ curl -XPOST 'http://localhost:9200/library/book?pretty&routing=books' -d
'{
    "author" : "Yuvraj Gupta",
    "title" : "Kibana Essentials",
    "pages" : 210
}'
```

在前面的示例中，routing 参数被指定为 books，程序会根据这一参数计算出一个哈希值。这本名为 Kibana Essentials 的书将被放入一个基于该哈希值所决定的分片中。

2. 获取文档 Get API

该 Index API 添加一个文档到索引中，Get API 用来根据文档的 id 来获取文档数据。下面获取 id 为 1 的文档：

```
$ curl -XGET 'http://localhost:9200/library/book/1?pretty'
```

使用 GET 方式来获取文档，并提供包含文档 id 的 URI。输出如下面的代码所示：

```
{
    "_index" : "library",
    "_type" : "book",
    "_id" : "1",
    "_version" : 1,
    "found" : true,
    "_source" : {
      "author" : "Ravi Kumar Gupta",
      "title" : "Test-Driven Javascript Development",
      "pages" : 240
    }
}
```

我们可以看到，输出信息包括与文档相关的信息，如索引的名称、类型、id、版本等。返回信息中 found 的值显示了文档及其 id 是否存在，source 对象包含实际已索引的文件。

为了检查文档是否存在，也可以使用 HEAD 方式。让我们看一下这个操作：

```
$ curl -XHEAD -i 'http://localhost:9200/library/book/2'
      HTTP/1.1 404 Not Found
      Content-Type: text/plain; charset=UTF-8
      Content-Length: 0
```

```
$ curl -XHEAD -i 'http://localhost:9200/library/book/1'
HTTP/1.1 200 OK
Content-Type: text/plain; charset=UTF-8
Content-Length: 0
```

获取文档时，搜索结果中表头的 source 展示了实际的文档内容。如果熟悉 Kibana 控制台并且尝试相同的命令操作，可能会看到一个报错。这是关于 Kibana 控制台的一个已知问题，可以访问网站 https://github.com/elastic/kibana/issues/9141 来了解。我们将在第 4 章 Kibana 界面中学习到 Kibana 控制台的有关内容。

有时可能不想检索整个内容，而只需要检索其中的少数字段，或者根本不需要 source 字段中的内容：

```
curl -XGET 'http://localhost:9200/library/book/1?_source=false'
```

前面的代码从输出信息中排除了 source 字段的内容。要跳过一些字段，例如，如果不想得到 pages 字段，应执行以下命令：

```
curl -XGET 'http://localhost:9200/library/book/1?_source_exclude=pages'
```

如果只需要 author 字段，并跳过所有其他字段，应执行以下命令：

```
curl -XGET'http://localhost:9200/library/book/1?_source_include=author'
```

可以同时利用_source_include 和_source_exclude。当一个文件有许多个字段，并且希望通过请求更少的字段来降低网络开销，这样的设置是有帮助的。为了理解这一点，将一本书连同类别一起插入到 Elasticsearch 索引中：

```
curl -XPUT "http://localhost:9200/library/book/4/_create?pretty" -d'
{
    "author" : "Ravi Kumar Gupta",
    "title" : "Test-Driven JavaScript Development",
    "pages" : 240,
    "category" : [
      {"name":"Technology","subcategory": "Javascript"},
      {"name":"Methodology","subcategory": "development"}
      ]
}'
```

现在如果想要得到文档类别，同时跳过子类别，可以使用下面的命令：

```
curl -XGET http://localhost:9200/library/book/4?
    _source_include=category&_source_exclude=*.subcategory
```

前面命令的响应如下：

```
{
        "_index": "library",
        "_type": "book",
"_id": "4",
          "_id": "4",
          "_version": 1,
          "found": true,
          "_source": {
            "category": [
              {
                "name": "Technology"
              },
              {
                "name": "Methodology"
              }
            ]
          }
      }
```

只需要使用_source_include，可以使用以下代码：

```
curl -XGET 'http://localhost:9200/library/book/1?_source=author'
```

返回信息将只包括 author，并排除了其他所有信息。

有时可能只需要 source 字段，在这种情况下，可以使用以下命令：

```
curl -XGET 'http://localhost:9200/library/book/1/_source'
```

也可以使用_source_include 和_source_exclude 来排除或包含 source 的字段。同样，使用相同的 uri 和 HEAD 方式，可以查看一个文档是否存在。

有时可能不关心文档属于哪一个类型，而希望用 id 来检索一个文档。对于这样的情况，可以指定类型为_all，将会得到任何类型中第一个匹配的 id 文档：

```
curl -XGET 'http://localhost:9200/library/_all/1/_source'
      curl -XGET 'http://localhost:9200/library/_all/1'
```

在创建索引时，可以显式地指定储存哪一个字段。例如，如果用以下映像创建一个 library 索引，应执行以下命令：

```
curl -XPUT "http://localhost:9200/library" -d'
      {
```

```
        "mappings": {
          "book": {
            "properties": {
              "author": {
                "type": "keyword",
                "store": true
              },
    "pages": {
                "type": "integer",
                "store": false
               }
            }
          }
        }
      }'
```

请注意，在 library 索引已经存在的情况下，执行前面的命令将抛出 index_already_exists_exception 异常。之后，添加一本 id 为 1 的一本书的信息，类似于以前添加的内容。现在，通过下面的命令传入 stored_field 参数时，程序将允许以数组的形式获得存储的字段：

```
curl -XGET "http://localhost:9200/library/book/1?stored_fields=author,pages"
```

我们将得到以下输出：

```
{
        "_index": "library",
        "_type": "book",
        "_id": "1",
        "_version": 1,
        "found": true,
        "fields": {
          "author": [
            "Ravi Kumar Gupta"
          ]
        }
      }
```

所有存储的字段将被返回为数组。

3. 删除文档 Delete API

Delete API 可从索引中删除文档，我们可以以 DELETE 方式来实现：

```
curl -XDELETE 'http://localhost:9200/library/book/1?pretty'
```

下面的 JSON 即为返回结果：

```
{
    "found" : true,
    "_index" : "library",
    "_type" : "book",
    "_id" : "1",
    "_version" : 2,
    "result" : "deleted",
    "_shards" : {
        "total" : 2,
        "successful" : 1,
        "failed" : 0
    }
}
```

如果文档存在，found 值将为 true，并且文档将被删除。

> 每一个写操作，包括删除操作，数据中 version 字段的值都会加 1。

4. 更新文档 Update API

Update API 可以更新文档。更新是通过编写的脚本来实现的。更新时，程序从索引中获取文档，在文档上运行脚本，并将数据重新存入索引。下面，在之前添加过的书中，增加一个类别：

```
curl -XPOST 'http://localhost:9200/library/book/1/_update?pretty' -d '{
          "script" : {
            "inline" : "ctx._source.category =\"category\"",
             "lang": "painless",
            "params" : {
            "category" : "Technical"
        }
    }
}'
```

> 早期版本的 Elasticsearch 默认情况下使用 Groovy 脚本语言。最新版本的 Elasticsearch(例如 5. x)使用了一个新开发的脚本语言,并将其嵌入到 Elasticsearch 中。这种语言被称为 Painless,它的语法与 Groovy 相似。本章后续部分进一步介绍这种语言。无论什么时候需要使用脚本,都可以简单地指定 lang 字段的值为 painless。我们将在整个章节中默认使用 Painless 脚本。

如果遇到一个文件丢失的错误,这可能是因为在之前的 Delete API 部分尝试操作时删除了 id 为 1 的书,此时只需尝试再添加一本书的信息。

在这里,如果不存在 category 字段,正在运行的 inline 脚本将添加此字段。如果存在 category 字段,则会更新这个字段:

```
"inline" : "ctx._source.category = category"
```

在上面的命令中,正在指定_source. category,它将从 param 对象的 param 类型中得到值。

ctx 映像包含 _index、_type、_id、_version、_routing、_parent、_timestamp、_ttl 和 _source 等变量。

运行前面的命令,将产生以下 JSON 格式的结果:

```
{
    "_index" : "library",
    "_type" : "book",
    "_id" : "1",
    "_version" : 2,
    "result" : "updated",
    "_shards" : {
      "total" : 2,
      "successful" : 1,
      "failed" : 0
    }
}
```

现在,id 为 1 的书的 category 字段值为 Technical。在这样的情况下,如果需要,可以通过使用 ctx. _source. remove()来删除一个字段:

```
$ curl -XPOST 'http://localhost:9200/library/book/1/_update?pretty' -d
'{
"script" : "ctx._source.remove("category")",
"lang" : "painless"
```

```
}
```

甚至可以设置一个条件,然后做一个更新:

```
"inline" : "ctx._source.category.contains(category) ?
ctx.op = "delete" : ctx.op = "none"
```

按照前面的命令,它将检查类别 category 的字段中是否包含一个叫作 Technical 的值,如果是,则删除该文档,否则什么也不做。

在这种情况下,也可以部分更新文档,只需要提供想要更新的字段即可:

```
$ curl -XPOST 'http://localhost:9200/library/book/1/_update?pretty' -d
'{
        "doc" : {
          "pages" : 250
        }
    }'
```

前面的代码只更新 id 为 1 的书的页数,在这种情况下这个值被合并。在 script 和 doc 同时出现在命令中时,doc 将会被忽略,只有 script 会运行。

如果 source 与旧文档不同,文档就会被重新索引。如果要更新文档,即使 source 没有变化,也可以通过设置 detect_noop 为 false 来实现:

```
$ curl -XPOST 'http://localhost:9200/library/book/1/_update?pretty' -d
      '{
        "doc" : {
          "pages" : 250
      },
        "detect_noop" : false
    }'
```

参考下面的代码:

```
$ curl -XPOST 'http://localhost:9200/library/book/3/_update?pretty' -d
'{
        "script" : {
          "inline" : "ctx._source.category = "category"",
          "lang": "painless",
          "params" : {
            "category" : "Technical"
              }
          },
           "upsert" : {
```

```
                "category" : "Technical"
            }
        }'
```

如果该文件不存在指定的 id(这种情况下是 3),那么 upsert 操作中的值将用于创建一个新的文档。也可以用 scripted_upsert,其中的脚本能够处理文档的初始化,这可以代替 upsert 操作。应该设置 scripted_upsert 属性为 true。

如果要将 doc 的内容作为新文档使用,则可以跳过添加 upsert 的步骤。此时,可以设置 doc_as_upsert 为 true,并且在操作中跳过添加 upsert 的部分:

```
$ curl -XPOST 'http://localhost:9200/library/book/5/_update?pretty' -d
'{
    "doc" : {
        "pages" : 250
    },
    "doc_as_upsert" : true
}'
```

2.3.1.2 多文档 API

下面的 API 支持对多个文档的操作。类似于单文档 API,它们也可以被划分为不同种类。

1. Multi-get API

顾名思义,这个 API 可一次获得多个文档。需要提供 index、type 和 id 的信息,其中 index 是必须提供的,而 type、id 可选。如果想获得 id 为 1 和 5 的文档,可以执行如下所示的命令:

```
$ curl -XGET 'http://localhost:9200/_mget?pretty' -d
      '{
         "docs" : [
        {
          "_index" : "library",
        "_id" : 1
         },
        {
          "_index" : "library",
          "_id" : 5
        }
       ]
```

```
    }'
```

以数组的形式查询 docs 的内容，在返回数据中可得到一系列文档。如果知道所有的文件都来自相同的索引，那么可以在 URI 中指定索引的名字，并且在命令中，使用'http://localhost：9200/library/_mgetpretty'，来代替 'http://localhost:9200/_mgetpretty'。

这样的情况同样适用于如下类型，可以在命令中使用'http://localhost：9200/library/book/_mgetpretty'，来代替'http://localhost：9200/_mgetpretty'。

按照上述方法更改了命令的写法之后，原有命令中的_id 部分就发生了重复。在这种情况下，可以进一步更改请求的写法，在命令中写入_ids 参数，它将包含_id 的数组。更改后的命令是这样的：

```
curl -XGET 'http://localhost:9200/library/_mget?pretty' -d  '{
        "ids" : [1, 5]
    }'
```

或者如下：

```
curl -XGET 'http://localhost:9200/library/book/_mget?pretty' -d '{
        "ids" : [1, 5]
     }'
```

如果不想指定类型的字段，则要么将类型的位置留空，要么使用_all 来代替。在这种情况下，程序会返回所有类型的文档中第一个匹配到指定 ids 的文档数据。

在结果集中，所有文档也将包含_source 字段。如果想跳过它或只想要其中的一部分字段，可以像单文档中的 Get API 那样，用_source、_source_include 和_source_exclude 作为 URL 参数。对于_source 的使用方法如下面代码所示：

```
curl -XGET 'http://localhost:9200/library/book/_mget?pretty' -d '{
        "docs" : [
        {
          "_index" : "library",
          "_id" : 1,
          "_source" : ["author"]
        },
        {
          "_index" : "library",
          "_id" : 5,
          "_source" : {
            "include" : ["author"],
            "exclude" : ["pages"]
              }
```

```
            }
        ]
}'
```

在 docs 中的第一部分,我们只想得到 author 的数据,并排除其他所有数据。第二部分在某种程度上也是相似的,我们想得到 author 的数据,并排除 pages 的数据。

类似地,也可以检索字段的信息,并指定一个字段的数组作为值。只有给定数组中的 stored_fields 数据将被返回。甚至可以在 URL 中设置一个默认的 stored_fields 返回列表,对每个文档也是如此:

```
$ curl 'http://localhost:9200/library/book/_mget?stored_fields=author'-d '{
  "docs" : [
  {
      "_id" : 1
  },
{
      "_id" : 5,
      "stored_fields":["pages"]
}
  ]
}'
```

前面的操作将针对未显式定义的字段来获得所有文档中 author 字段的数据。对于 id 为 5 的文档,只有 pages 字段被映射为已存储字段(stored field)时,才能得到返回的 pages 字段。

2. Bulk API

有时需要对大量文档进行批操作,而 Bulk API 将会在这种情况下起作用。可以通过使用这种 API 来索引、创建、删除和更新文档,可以使用如下所示的 API:

```
/_bulk
/index/_bulk
/index/type/_bulk
```

2.3.2 有关搜索的 API

这些 API 可以帮助用户搜索一个或多个索引,下面将更详细地探讨这些 API。

2.3.2.1 搜索 API

这个 API 可搜索一个或多个索引,以及零个或多个类型。搜索操作可以通过两种方式

进行：在搜索 URI 中写入查询参数，或者在请求中使用**领域特定语言**（**DSL**）查询。操作将返回命中的文档数量，这一数量即为查询结果的数量。

在这种方法中，可使用“q=”来指定搜索参数。例如，如果想搜索作者名字包含 gupta 的所有图书，可以使用 GET 方式查询：

```
$ curl -XGET 'http://localhost:9200/library/_search?q=author:gupta&pretty'
```

执行这个命令将返回以下 JSON 格式的结果：

```
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
},
  "hits" : {
    "total" : 2,
    "max_score" : 0.37158427,
    "hits" : [ {
      "_index" : "library",
      "_type" : "book",
      "_id" : "AVSPSXXDxiIiqkaLJfTy",
      "_score" : 0.37158427,
      "_source" : {
        "author" : "Yuvraj Gupta",
        "title" : "Kibana Essentials",
        "pages" : 210
      }
  },   {
      "_index" : "library",
      "_type" : "book",
      "_id" : "1",
      "_score" : 0.2972674,
      "_source" : {
        "author" : "Ravi Kumar Gupta",
        "title" : "Test-Driven JavaScript Development",
        "pages" : 250,
        "category" : "Technical"
      }
```

```
        } ]
      }
    }
```

正如我们所看到的,得到了 Kibana Essentials 和 Test-Driven Javascript 这两个命中的结果。

与 POST/PUT 操作不同,GET 操作的结果中将会返回_shards 部分的总分片数。

在前面的调用过程中,没有指定任何类型。如果要指定类型,可以调用以下命令:

```
$ curl -XGET 'http://localhost:9200/library/book/_search?q=author:gupta&pretty'
```

如果有多个索引或多个类型,可以使用以下方法进行搜索:

```
$ curl -XGET 'http://localhost:9200/library,users/_search?q=author:gupta&pretty'
$ curl-XGET
'http://localhost:9200/library/book,journal/_search?q=author:gupta&pretty'
```

> 如果在搜索时没有找到索引,程序会抛出 index_not_found_exception 异常。

如果想搜索所有的索引和类型,可以跳过在 URI 中指定索引和类型的步骤:

```
$ curl -XGET 'http://localhost:9200/_search?q=author:gupta&pretty'
```

可以在其中指定任意数量的索引和类型(用逗号分隔)。

2.3.2.2 搜索分片 API

这个 API 可在执行搜索时获取索引和分片。在搜索时,可以提供逗号分开的索引和类型。为了理解这一点,让我们看看下面的 URI:

```
$ curl -XGET 'http://localhost:9200/library/_search_shards?pretty'
```

执行这条命令,将返回所有用于搜索和索引的分片。我们可以提供一个 routing 的值:

```
$ curl -XGET 'http://localhost:9200/library/_search_shards?pretty&routing=
gupta'
```

现在将得到一个简短的列表,当然也可以指定多个 routing 值(用逗号分隔)。

2.3.2.3 多搜索 Multi-search API

这种 API 可一次执行多个搜索操作,我们需要提供一个包含搜索请求头部和主体部分的文件。请求头的部分指定索引、要搜索的类型、search_type、首选项和 routing。主体部分包含查询、来源、大小、聚合等。例如,思考以下内容:

```
{"index" : "library", "type" : "book" }
{"query" : {"term" :{"author" : "gupta"}} }
```

在上述的代码中，第一行是头部，第二行是主体部分。在文件中有尽可能多的键值对。一旦准备好了查询，即可使用 GET 方式来操作。假设将所有查询命令放在一个名为 queries 的文件中：

```
curl - XGET ' http://localhost: 9200/_ msearch? pretty ' - - data - binary @
queries; echo
```

使用--data-binary，可以指定包含所有头部和主体的文件，@queries 是文件名。

2.3.2.4 计数 API

正如其名称所示，这个 API 可以用来获取查询结果的匹配数量，可以使用_count 参数来获得匹配结果的数量：

```
curl -XGET 'http://localhost:9200/library/book/_count?pretty' -d '{
    "query" : {
      "term" : {"author" : "gupta"}
    }
}'
```

这一操作的返回结果如下：

```
{
    "count" : 2,
    "_shards" : {
        "total" : 5,
        "successful" : 5,
        "failed" : 0
    }
}
```

结果显示在这五个分片之中，有两个匹配到了文档。

2.3.2.5 验证 API

如果正在对敏感数据进行查询，或者要花费太多的时间，以及其他原因，就可以先对查询进行验证，然后再执行查询。这个 API 可在执行查询之前，验证这个查询是否有效。

为了使用这个 API，可以使用/_validate/query：

```
$ curl -XGET'http://localhost:9200/library/book/_validate/query?pretty' -d '{
```

```
      "query" : {
        "term" : {"author" : "gupta"}
        }
  }'
```

该操作将针对验证的字段返回 true 或 false 的结果。

2.3.2.6 结果解释 API

这个 API 解释了查询指令和特定文档的分值计算，可以使用/_explain 来实现这个目的。在这个操作中，需要提供一个索引和一个类型：

```
$ curl -XGET 'http://localhost:9200/library/book/1/_explain?pretty' -d
    '{
      "query" : {
        "term" : {"author" : "gupta"}
     }
    }'
```

在以上命令中，查询了 id 为 1 的图书，查询中指定了作者的名字为 gupta。

2.3.2.7 分析 Profile API

有时查询可能会非常缓慢，这个 API 可以帮助我们了解底层网络传输的细节，找到哪个查询组件花费了多长时间，以便分析和执行操作。此 API 仍然是在 ES 5.x 版本中的实验性功能。为了在调用 API 过程中启用 Profile，可以执行以下代码：

```
$ curl -XGET "http://localhost:9200/library/_search?q=author:gupta&pretty" -d'
    {
      "profile": true
    }'
```

在结果中，通常会看到基本信息及信息的命中数量，但是现在也会有详细的指标分析信息。在通过查询搜索到的索引中，上述命令会对其中每个分片的各项指标进行分析。这里面包含完整的查询执行细节。

X-Pack 插件中也提供了一个这样的分析器，可参见第 9 章的相关内容。

2.3.2.8 字段统计信息 API

这个 API 提供了针对一个或多个字段的统计信息，可以使用_field_stats 来获取。例如：

```
$ curl -XGET 'http://localhost:9200/_field_stats?pretty&fields=pages'
```

默认情况下，上述操作会以 cluster(集群)级别执行。与命令中的 fields 类似，还可以使用一个 level 来定义其为 cluster(集群)级别还是 indices(索引)级别。这个结果展示了分片、计数、最小值、最大值、密度等诸多指标。

```
{
  "_shards": {
   "total": 5,
   "successful": 5,
   "failed": 0
  },
  "indices": {
   "_all": {
    "fields": {
     "pages": {
      "type": "integer",
      "max_doc": 2,
      "doc_count": 2,
      "density": 100,
     "sum_doc_freq": -1,
     "sum_total_term_freq": 2,
     "searchable": true,
     "aggregatable": true,
     "min_value": 210,
     "min_value_as_string": "210",
     "max_value": 240,
     "max_value_as_string": "240"
    }
   }
  }
 }
}
```

如果有任何值为－1，则意味着该字段的度量对于一个或者多个分片不可用。

这个 API 有助于找出最小值、最大值、词项计数等。它也是实验性的，可以在今后的发行版本中删除。

2.3.3 有关索引的 API

索引 API 可以帮助我们构建和管理索引、设置、映像、别名和模板。本节将介绍它提供

的功能。

2.3.3.1 管理索引

本节介绍创建、更新、删除索引和设置的方法。

1. 创建一个索引

要创建索引，可使用带有 curl 的 PUT 方式。在创建索引时，可以指定一些设置，例如分片、映像、别名等。要创建具有所有默认设置的索引，可以使用以下内容：

```
$ curl -XPUT 'localhost:9200/library'
```

执行该命令将尝试创建一个名为 library 的索引，如果可以完成的话，输出将是：

```
{"acknowledged":true}
```

此操作将在 library 索引中创建五个主分片和五个副分片（一个主分片对应一个副分片），可以更改映像和别名的设置。让我们看一个更复杂的例子：

```
$ curl -XPUT 'http://localhost:9200/library' -d '{
    "settings" : {
      "number_of_shards" : 2,
      "number_of_replicas" : 1
    }
 }'
```

上述操作将为索引创建两个主分片，每个主分片拥有一个副分片。

> 一个索引被创建后，如果尝试重复创建，那么程序可能会抛出 index_already_exists_exception 错误异常。

2. 检查索引是否存在

可以在 curl 命令中以 HEAD 方式来检查一个索引是否存在。例如，检查 library 索引是否存在：

```
$ curl -XHEAD -i 'http://localhost:9200/library'
HTTP/1.1 200 OK
Content-Type: text/plain; charset=UTF-8
Content-Length: 0
```

如果状态码为 404，则表示索引不存在。

3. 获取索引信息

可以使用以下方式获得有关索引的信息：

```
$ curl -XGET 'localhost:9200/library?pretty'
```

执行该命令将显示关于 library 的索引信息，其中包括别名、映像、设置和 warmer。设置包含分片信息、创建时间、版本等。

```
{
  "library": {
    "aliases": {},
    "mappings": {
      "book": {
        "properties": {
          "author": {
            "type": "keyword",
            "store": true
          },
            "pages": {
            "type": "integer"
          },
          "title": {
            "type": "keyword",
            "store": true
          }
        }
      }
  },
    "settings": {
      "index": {
        "creation_date": "1484693593662",
        "number_of_shards": "5",
        "number_of_replicas": "1",
        "uuid": "kW1tscieT6OaWClNnhguAQ",
        "version": {
          "created": "5010199"
        },
        "provided_name": "library"
      }
    }
  }
}
```

如果没有可用的信息，就会返回一个空数组，在别名中也是一样的结果。

> ℹ如果尝试对不存在的索引查询信息，那么程序可能会抛出 index_not_found_exception 异常。

4. 管理索引设置

如果想得到的只是设置的内容，则可以在命令中使用/{index}/_settings：

```
$ curl -XGET 'http://localhost:9200/library/_settings?pretty'
```

{index}的部分也可以指定多个索引，各个索引名称之间以逗号分隔。

可以使用通配符来匹配索引，还可以在创建索引后更新设置。可以以 PUT 方式使用相同的{index}/_settings，来完成更新，设置将会是动态更新的。如果不指定任何内容，程序会更新全部索引的设置：

```
$ curl -XPUT 'http://localhost:9200/library/_settings?pretty' -d '{
        "index" : {
        "number_of_replicas" : 2
      }
      }'
```

上述操作将在 library 索引上为每一个主分片设置两个副分片。

到目前为止，我们已经了解了创建、编辑、打开和关闭操作。除此之外，可以通过检查统计信息、分片存储信息、恢复信息和段信息来对索引进行监控。

5. 获得索引的统计信息

此操作可以获得有关索引的统计信息。它会显示文件、索引大小、索引状态、搜索、字段数据、更新、合并、请求缓存、刷新、建议以及其他统计信息。为了获得索引的统计信息，可以执行以下命令：

```
$ curl -XGET 'http://localhost:9200/library/_stats?pretty'
```

可以提供以逗号分隔的多个索引，以便一次获得多个索引的统计信息；还可以指定要获取哪些特定的统计信息，如下面的命令所示：

```
$ curl -XGET 'http://localhost:9200/library/_stats/docs,search?pretty'
```

6. 获取索引分段

在一个或多个索引上使用_segment，可以得到一个底层的有关索引分段的信息：

```
$ curl -XGET 'http://localhost:9200/library/_segments?pretty'
```

7. 获取索引恢复信息

这个 API 提供了关于索引分片的恢复信息。这为索引的每一个分片提供了一个非常细致的视角。可以在一个或多个索引上使用_recovery 来获取索引的恢复信息：

```
$ curl -XGET 'http://localhost:9200/library/_recovery?pretty'
```

8. 获取分片存储信息

这个 API 有助于获取一个或多个索引的分片存储信息。可以使用_shard_stores 来得到分片的存储信息：

```
curl -XGET 'http://localhost:9200/library/_shard_stores?pretty'
```

这将影响所有的分片，下面是其中一个这样的分片：

```
"0" : {
    "stores" : [ {
        "re4j45-yTp6VZgMN7dP2Sg" : {
            "name" : "xB20COp",
            "ephemeral_id" : "SK9sdP0nRPuCKs51VtvNqg",
            "transport_address" : "127.0.0.1:9300",
            "attributes" : { }
        },
        "allocation_id" : "9I_4rywATD6CV8lqvFIFaA",
        "allocation" : "primary"
    }
  ]
}
```

返回的结果中，显示了许多存储信息，包括节点名称、地址、属性、版本和分配类型。如果只运行了一个节点，则不会获得有关副本分片的信息。

9. 索引别名

别名与 UNIX 操作系统中的别名相似。在 UNIX 操作系统中，它们是用于命令的，而在这里是用于索引的。

我们可以为单个或多个索引创建别名。别名创建后，可在 API 调用中使用它来代替索引名称，而 Elasticsearch 将用实际索引名称替换别名然后执行操作。有许多索引时，别名是有用的，可以根据它对索引分组。

例如，已经搭建了生产服务器，其中含有数据库、网络和应用服务器程序，同时也已经建立起 Elastic Stack 从服务器中收集日志，并存入 Elasticsearch。现在还有其他索引，但是希望对所有日志相关的索引执行一些操作。这时可以用一个别名指代它们，例如 prod_logs。

以后无论想执行什么操作，都可以通过简单地使用 prod_logs 替代这些与日志相关的索引名而不是输入所有的索引名。可以使用_aliases 创建别名。可以为索引添加和删除别名，如下面的代码段所示：

```
$ curl -XPOST 'http://localhost:9200/_aliases' -d '{
    "actions" : [
        { "add" : { "index" : "library", "alias" : "alias1" } },
        { "remove" : { "index" : "library", "alias" : "alias1" } }
    ]
}'
```

可以指定尽可能多的操作。调用的第一个操作是在 library 索引中添加一个别名 alias1，第二个操作是将其删除。

> 别名不能和索引名称相同。

10. 映像

根据文档里面的数据将其添加给一个索引时，Elasticsearch 会创建映像。这些映像能帮助我们识别和定义一个字段是否为全文索引的字段、数字字段或日期字段。

在 Elasticsearch 中，为了将文档在索引中进行逻辑分组，每个指标都有一个或多个映像类型。每一个映像类型都有 meta-fields（元字段，如_index、_type、_id、_source）和普通内容字段列表，每个字段都有数据类型。这些数据类型可以是 string（字符串）、long（长整型）、double（双精度浮点数）、date（日期）、boolean（布尔型）、ip（IP 地址）、object（对象）、nested（嵌套）以及与地理位置有关的特殊类型，如 geo_point、geo_shape 等。

可以通过_mapping 来获取索引的映像：

```
$ curl http://localhost:9200/library/_mapping?pretty
```

如果有多个类型，而且只想针对特定类型获得映像，可以使用{index}/_mapping/{type}：

```
$ curl http://localhost:9200/library/_mapping/book?pretty
```

执行此命令将输出一个包含所有字段的属性表：

```
{
  "library": {
    "mappings": {
      "book": {
        "properties": {
```

```
                "author": {
                  "type": "text",
                  "fields": {
                    "keyword": {
                      "type": "keyword",
                      "ignore_above": 256
                    }
                  }
                },
                "pages": {
                  "type": "long"
                },
                "title": {
                  "type": "text",
                  "fields": {
                    "keyword": {
                      "type": "keyword",
                      "ignore_above": 256
                    }
                  }
                }
              }
            }
          }
        }
      }
```

其中的 book 类型包含这三个字段，对于每个字段，都有一个与其他相关属性一起定义的类型。

可以通过使用{index}/_mapping/{type}/field/{fields}来获取特殊字段的映像，在命令中可以指定以逗号分隔的多个字段：

```
$ curl localhost:9200/library/_mapping/field/title?pretty
```

可以以 HEAD 方式来检查类型是否存在：

```
$ curl -XHEAD -i 'http://localhost:9200/library/_mapping/book'
HTTP/1.1 200 OK
Content-Type: text/plain; charset=UTF-8
Content-Length: 0
```

状态码 200 表示该类型存在。如果不存在，则状态码为 404。

可以使用 Put Mappings API 来添加映像，允许添加一个新类型或新字段。

若要向索引添加新类型，例如将 paper 添加到 library 索引中，可以执行以下命令：

```
$ curl -XPUT 'http://localhost:9200/library/_mapping/paper' -d'
        {
              "properties": {
                "abstract": {
                  "type": "string"
                }
              }
        }'
```

如果成功创建，则执行此操作的运行结果为 true。这将创建一个 paper 类型，其中包含一个 abstract 字段。

11. 关闭、打开和删除索引

有时出于维护或恢复的目的，可能希望停止对一个特定索引的读/写操作。当索引关闭时，除了保持元数据之外，集群上不会执行任何操作。可以分别使用/{index}/_close 和/{index}/_open 来关闭/打开索引。为此，以 POST 方式来执行 curl 命令：

```
$ curl -XPOST 'http://localhost:9200/library/_close'
$ curl -XPOST 'http://localhost:9200/library/_open'
```

如果根本不需要索引，也可以以 DELETE 方式删除它：

```
$ curl -XDELETE 'http://localhost:9200/library'
```

这将从集群中删除索引，就像它从未出现过一样。还可以使用通配符匹配来删除多个索引，例如删除所有名称为 textXXX 的索引：

```
$ curl -XDELETE 'http://localhost:9200/test*'
```

这将查看众多索引中是否包括某些名称以 test 开头的索引，如果有，则将其删除。如果不存在名称为 test 的索引并且此时正在使用通配符，也不会得到任何错误，而是如下的确认信息：

```
{
    "acknowledged": true
}
```

如果在没有通配符的情况下删除一个索引，同时索引不存在，那么程序将会抛出 index_not_found_exception 异常。

2.3.3.2 其他操作

这个 API 支持的操作还有很多，例如清除缓存、升级 Elasticsearch 索引、强制合并、更新索引和刷新等：

- **清除缓存**：_cache/clear 可以清除一个或多个索引的缓存：

```
$ curl -XPOST "http://localhost:9200/library/_cache/clear"
```

- **Flush**：_flush 通过清除事务日志以及将数据转移到索引中进行存储的方式，释放一个或多个索引所占用的内存空间：

```
$ curl -XPOST "http://localhost:9200/library/_flush"
```

- **更新索引 Refresh**：提供_refresh 来更新一个或多个索引：

```
$ curl -XPOST "http://localhost:9200/library/_refresh"
```

- **软件更新 Upgrade API**：将 Elasticsearch 索引从旧版本升级到新版本，可以用_upgrade 升级一个或多个索引。这个过程通常需要一些时间：

```
$ curl -XPOST 'http://localhost:9200/library/_upgrade?pretty'
```

程序将返回以下输出：

```
{
    "_shards": {
        "total": 15,
        "successful": 10,
        "failed": 0
    },
    "upgraded_indices": {
        "library": {
            "upgrade_version": "5.1.1",
            "oldest_lucene_segment_version": "6.3.0"
        }
    }
}
```

正如我们所见，library 索引已经被成功升级，其升级后的版本已经发布。

2.3.4 Cat API

Cat API 以可读的格式（而非 JSON 格式）输出节点、索引、字段、任务和插件的信息，也

可以可视化表格输出在控制台上的方式。

在第 8 章 Elasticsearch API 的 Cat API 中，将学习更多关于 Cat API 的内容。

2.3.5 Cluster API

Cluster API 可使我们了解集群的状态、健康程度、统计数据、节点统计数据和节点信息。在第 8 章 Elasticsearch API 的集群 API 中将学习有关集群 API 的内容。

2.4 Query DSL

这种方式需要像调用 Document API 那样，指定一个带有 uri 的请求体。对于前面按作者搜索的查询，可以重写为如下内容：

```
$ curl -XGET 'http://localhost:9200/library/book/_search?pretty' -d '{
          "query" : {
            "term" : {"author" : "gupta"}
          }
        }'
```

此查询将返回相同的结果。任何使用“q=”定义的查询参数，都可在 term 中定义。要了解有关 Query DSL 的更多信息，可以参阅 https://www.elastic.co/guide/en/elasticsearch/reference/5.1/query-dsl.html。

2.5 聚合

聚合(Aggregations)框架是 Elasticsearch 中非常重要的一部分。顾名思义，这个框架能聚合和生成关于搜索查询结果的分析信息。聚合有助于更好地了解数据。例如，在 library 索引示例中，可以得到一些答案：特定的某一年中有多少本书，书是关于哪种技术的，平均每年有多少本书以及更多的其他信息。

当从面板上了解系统的数据时，聚合就能显现出其优越性了。通常系统面板上会有图表形式的聚合数据。后面的章节中也将使用聚合，这些聚合信息将帮助 Kibana 生成有用的可视化内容。

核心的聚合有两种类型：测度(Metrics)和桶(Buckets)聚合。这一部分将学习这些内容。

2.5.1 Buckets 聚合

这种聚合方式是基于某种标准创建文档的桶。它也可以包含子聚合。在这一节中，我

们将学习子聚合。

要理解基于桶的聚合，首先添加另一个索引 stones 和一个名为 diamonds 的类型。该数据集可通过访问下列地址获得：https://vincentarelbundock.github.io/Rdatasets/datasets.html。

为方便起见，所使用的数据集也与本章内容一起打包，其代码文件也可以获取。如果想尝试未修改的数据集，可以在这个页面中获取：https://vincentarelbundock.github.io/rdatasets/csv/ecdat/diamond.csv。数据集的示例如下表所示：

重量/克拉	切割	颜色	净度	价格/$
0.23	理想	透明无色(E)	细小杂质(SI2)	326
0.21	优质	透明无色(E)	细小杂质(SI1)	326
0.23	好	透明无色(E)	非常细小杂质(VS1)	327
0.29	优质	接近无色(I)	非常细小杂质(VS2)	334
0.31	好	接近无色(J)	细小杂质(SI2)	335

此表中的一行显示了有关钻石的数据，我们得到了重量、切割、颜色、净度和价格这些信息。这里还有其他的字段，但是为了简单起见，省略了那些字段。

将数据存入 Elasticsearch 的 Logstash 配置文件，内容如下所示：

> 第 3 章探索 Logstash 及其组件，将会涉及 Logstash 的内容。

```
input{
   file{
     path =>"/opt/elk/datasets/diamonds.csv"
     start_position =>"beginning"
   }
}
   filter{
      csv{
        columns =>
                  ["caret", "cut", "colour", "clarity", "depth",
"table", "price", "x", "y", "z"]
        separator =>","
      }
     mutate {
            convert =>["caret","float"]
            convert =>["depth","float"]
```

```
                convert => ["table","integer"]
                convert => ["price","integer"]
                convert => ["x","integer"]
                convert => ["y","integer"]
                convert => ["z","integer"]
        }
}
output {
            elasticsearch {
              index => "stones"
              document_type => "diamond"
              hosts => "localhost"

        }
}
```

在前面的配置中，我们添加了一个 mutate 代码段来将字段转换为适当的类型。如果不转换，Elasticsearch 就将所有字段设置为字符串类型。另外，请注意我们已经添加了一个名为 stones 的索引（如果这个索引不存在，将自动为这个名称创建一个新索引）和 diamond 的文档类型。这将在 Elasticsearch 上创建一个名称为 stones 的索引，并为这个索引创建类型 diamond。如果索引和类型已经存在，那么数据将被添加进去。没有在这里为 Elasticsearch 指定端口，将使用默认端口 9200。要使用此配置运行 Logstash，请执行以下命令（假设配置文件被命名为 logstash-diamond. conf 并且放置在 conf 目录下）：

```
$ ./bin/logstash -f conf/logstash.diamonds.conf
```

最后一个命令将创建一个索引，在这个索引上将尝试使用聚合。在搜索查询中，添加一个聚合，代码如下：

```
"aggs" : {
    "<aggregation-name>" : {
        "<aggregation-type>" : {
            "field" : "<field-name>"
        }
    }
}
```

这是定义聚合的方式——使用 aggs 参数（使用 aggregations 代替 aggs 也是可以的）。除了 aggs 参数，需要为聚合设置一个名字、一个要使用的类型。最后，需要提供一个对数据进行聚合的字段。

让我们创建第一个 bucket 聚合。在这一点上，希望按净度(clarity)这一指标将钻石放入桶中：

```
$ curl -XGET 'http://localhost:9200/stones/diamond/_search?pretty'
       -d '{
        "aggs" : {
          "diamonds_by_clarity" : {
            "terms" : {
              "field" : "clarity"
            }
          }
        }
      }'
```

可能会得到一个 illegal_argument_exception 异常，且在默认情况下，Fielddata 在文本字段中被禁用。此时需要对 clarity 字段设置 fielddata=true，通过逆向解析倒排索引，将 fielddata 加载到内存中。注意，这将消耗大量的内存空间。要启用字段 fielddata，例如对 clarity 字段执行此操作，请执行以下命令：

```
$ curl -XPUT "http://localhost:9200/stones/_mapping/diamond" -d'  {
  "properties": {
          "clarity": {
            "type":     "text",
            "fielddata": true
          }
        }
    }'
```

这将创建净度的 buckets，其输出如下：

```
        "aggregations" : {
          "diamonds_by_clarity" : {
            "doc_count_error_upper_bound" : 0,
            "sum_other_doc_count" : 0,
            "buckets" : [ {
              "key" : "si1",
              "doc_count" : 13065
            }, {
              "key" : "vs2",
              "doc_count" : 12258
            }, {
              "key" : "si2",
```

```
            "doc_count" : 9193
          }, {
            "key" : "vs1",
            "doc_count" : 8171
          }, {
            "key" : "vvs2",
            "doc_count" : 5066
          }, {
            "key" : "vvs1",
            "doc_count" : 3655
          }, {
            "key" : "if",
            "doc_count" : 1790
          }, {
              "key" : "i1",
            "doc_count" : 741 }
          }  ]
}
}
```

可以看到，每一个不同的净度值都有唯一的 bucket，净度值从之前“键值对”中的“值”变为“键”，每个 bucket 中都以 doc_count 来表示所有包含该净度值的文档的数量统计。

增加一个 metric 聚合作为子聚合，可增加它的复杂性：

```
$ curl -XGET 'http://localhost:9200/stones/diamond/_search?pretty' -d
      '{
        "aggs" : {
          "diamonds_by_clarity" : {
            "terms" : {
              "field" : "clarity"
},
            "aggs" : {
              "max_price" : {
                "max" : {
                  "field" : "price"
                }
              }
             }
          }
        }
    }'
```

要添加子聚合，只需在创建好的聚合中再增加一个 aggs 参数。其中的 max 是一个 metric 聚合，这些将在下一节学习到。让我们分析一下这一输出：

```
"buckets" : [ {
        "key" : "si1",
        "doc_count" : 13065,
        "max_price" : {
          "value" : 18818.0
        }
      }, {
        "key" : "vs2",
        "doc_count" : 12258,
        "max_price" : {
          "value" : 18823.0
        }
      },
  ...
]
```

现在每个 bucket 都将包含文档中的 max_price。

例如，使用价格范围去分析数据。可以看到，价格最大值小于 19 000 $，据此定义价格为 0～5000 $、5000～10 000 $、10 000～14 000 $、14 000～16 000 $、16 000～19 000 $。这次使用的聚合是范围(range)聚合：

```
$ curl -XGET 'http://localhost:9200/stones/diamond/_search?pretty' -d
      '{
        "aggs" : {
          "price_ranges" : {
            "range" : {
              "field" : "price",
              "ranges" : [
                { "to" : 5000 },
                {"from" : 5000, "to" : 10000 },
                {"from" : 10000, "to" : 14000 },
                {"from" : 14000, "to" : 16000 },
                {"from" : 16000, "to" : 19000 }
              ]
            }
          }
        }
    }'
```

还需要定义这个聚合的范围。现在，输出中将会包含刚刚定义好范围的 buckets：

```
"aggregations" : {
  "price_ranges" : {
    "buckets" : [ {
      "key" : "*-5000.0",
      "to" : 5000.0,
      "doc_count" : 39212
    }, {
      "key" : "5000.0-10000.0",
      "from" : 5000.0,
    "to" : 10000.0,
      "doc_count" : 9504
    }, {
      "key" : "10000.0-14000.0",
      "from" : 10000.0,
      "to" : 14000.0,
      "doc_count" : 3064
    }, {
      "key" : "14000.0-16000.0",
      "from" : 14000.0,
      "to" : 16000.0,
      "doc_count" : 1017
}, {
"key" : "16000.0-19000.0",
      "from" : 16000.0,
      "to" : 19000.0,
      "doc_count" : 1142
    }   ]
  }
}
```

可以看到文档按定义的范围进行了划分。

做更多的聚合之前，先把更多的数据存入 Elasticsearch，并为 library 库索引添加一个名为 movies 的新类型。要使用 IMDB 的数据集，可以从这个网址下载：https://vincentarelbundock.github.io/Rdatasets/csv/ggplot2/movies.csv。

已经删除了 CSV 文件中第一列的 S. No. 字段，然后使用 Logstash 加载其余数据。Logstash 加载的配置文件内容如下所示：

```
input{
file{
```

```
path =>"/ opt/elk/datasets/movies.csv"
          start_position =>"beginning"
        }
      }
filter{
csv{
columns =>
               ["title", "year", "length", "budget", "rating", "votes",
  "r1votes", "r2votes", "r3votes", "r4votes", "r5votes", "r6votes",
  "r7votes", "r8votes", "r9votes", "r10votes", "mpaaRating", "action",
  "animation", "comedy", "drama", "documentary", "romance", "short"]
separator =>","
           }
mutate {
convert =>["year","integer"]
convert =>["budget","integer"]
convert =>["votes","integer"]
convert =>["rating","integer"]
convert =>["action","integer"]
convert =>["animation","integer"]
convert =>["comedy","integer"]
convert =>["drama","integer"]
convert =>["documentary","integer"]
convert =>["romance","integer"]
convert =>["short","integer"]
        }
  }
output {
elasticsearch {
index =>"library"
               document_type =>"movies"
hosts =>"localhost"
         }
     }
```

可以像上面 stones 那个例子一样运行这个配置。唯一的变化是，索引名称与 library 索引相同，并将 document_type 设置为 movies：

```
$ ./bin/logstash -f conf/logstash.movies.conf
```

2.5.2 Metrics 聚合

进行索引时，一些数字或值被提取出来，Metrics 聚合从文档中计算这些指标，数值指标

分为单值和多值。在 Metrics 聚合中有许多可用的测度聚合，在这一节中，将学习一些重要的 Metrics 聚合。

2.5.2.1 Avg 平均值聚合

这是一个单值聚合，它用数值去计算聚合文档中相应测度的平均值。这些值可以取自特定的字段，也可以是某些脚本运行的结果，这是一个单值测度。例如，如果希望得到 votes 字段的平均数，可以执行以下命令：

```
$ curl -XGET 'http://localhost:9200/library/movies/_search?pretty' -d
    '{
      "aggs" : {
        "avg_votes" : { "avg" : { "field" : "votes" } }
    }
  }'
```

执行后将返回一些聚合结果的记录：

```
{
        ...
        "aggregations" : {
              "avg_votes" : {
              "value" : 632.1034394774443
              }
          }
    }
```

2.5.2.2 Min 最小值聚合

这是一个单值聚合，可以返回从聚合文档中提取的数据的最小值。例如，要找到 rating 字段中的最小值，可执行以下命令：

```
{
    "aggs" : {
        "min_rating" : { "min" : { "field" : "rating" } }
    }
}
```

2.5.2.3 Max 最大值聚合

这是一个单值聚合，可以返回从聚合文档中提取的数据的最大值。例如，要找到 rating

字段中最大的数据值，可执行以下命令：

```
{
    "aggs" : {
        "max_rating" : { "max" : { "field" : "rating" } }
    }
}
```

2.5.2.4 Percentiles 百分比聚合

这是一个在数值字段上生成百分数的多值聚合数值。例如，在 rating 字段上得到百分比。要获得该百分比，可以执行以下命令：

```
{
        "aggs" : {
          "rating_percentiles" : {
            "percentiles" : { "field" : "rating" }
          }
        }
    }
```

返回的结果如下所示：

```
{
    ...
        "aggregations" : {
          "rating_percentiles" : {
            "values" : {
                    "1.0" : 1.0,
                    "5.0" : 3.0,
                    "25.0" : 5.0,
                    "50.0" : 6.0,
                    "75.0" : 7.0,
                    "95.0" : 8.0,
                    "99.0" : 9.0
            }
        }
    }
}
```

2.5.2.5 Sum 求和聚合

这是一个单值聚合，它可以返回到从字段中提取数值的总和。对于数据集，如果想找到

喜剧电影（在 comedy 字段进行计算）的总的数量，可以执行如下命令：

```
$ curl -XGET 'http://localhost:9200/library/movies/_search?pretty' -d
      '{
        "aggs" : {
          "total_comedy_movies" : { "sum" : { "field" : "comedy" } }
        }
      }'
```

执行上述命令将返回以下结果：

```
"aggregations" : {
    "total_comedy_movies" : {
        "value" : 17271.0
    }
}
```

2.5.2.6 Value_count 计数聚合

该统计是一个单值聚合，它可以获取搜索到的文档和字段的数量计数值。例如，要获得 2000 年发布的电影的数量，可以执行以下命令：

```
$ curl -XGET'http://localhost:9200/library/movies/_search?pretty&q=year:2000'
        -d '{
        "aggs" : {
          "total_rated_movies" : { "value_count" : { "field" : "rating" } }
      }
   }'
```

2.5.2.7 Cardinality 聚合

Cardinality（基数）聚合也是一个单值聚合，可以将其看作类似关系型数据库中的 distinct 查询。假设想得到 movies 索引中不重复的 clarity 统计数量，可以使用这个聚合：

```
$ curl 'http://localhost:9200/library/movies/_search?pretty' -d '{
        "aggs" : {
          "years" : {
            "cardinality" : {
              "field" : "year"
            }
          }
      }
```

```
}'
```

执行该命令将得到索引中不重复的 clarity 计数：

```
"aggregations" : {
    "years" : {
        "value" : 68
    }
}
```

2.5.2.8 Stats 聚合

Stats(统计)聚合是一种多值聚合，计算之后返回该字段的 min、max、sum、count 和 avg 等统计数据。要计算 votes 的统计数据，执行以下操作：

```
$ curl -XGET 'http://localhost:9200/library/movies/_search?pretty' -d
    '{
        "aggs" : {
            "stats_votes" : { "stats" : { "field" : "votes" } }
        }
}'
```

执行上述命令将返回以下输出：

```
"aggregations" : {
      "stats_votes" : {
            "count" : 58787,
            "min" : 5.0,
            "max" : 149494.0,
            "avg" : 629.4601357442972,
            "sum" : 3.7004073E7
    }
}
```

2.5.2.9 扩展 Stats 聚合

这种多值聚合是一种扩展的 Stats 聚合。除了最小值、最大值、总和、计数和均值之外，它还增加了 sum_of_squares(平方和)、variance(方差)、std_deviation(标准差)和 std_deviation_bounds(标准差范围)等统计数据。例如，计算 votes 中的扩展统计数据：

```
$ curl -XGET 'http://localhost:9200/library/movies/_search?pretty' -d
      '{
```

```
            "aggs" : {
              "extended_stats_votes" : { "extended_stats" : { "field" : "votes" }
            }
          }
        }'
```

执行上述命令将返回以下输出：

```
"aggregations" : {
    "extended_stats_votes" : {
        "count" : 58787,
        "min" : 5.0,
        "max" : 149494.0,
        "avg" : 629.4601357442972,
        "sum" : 3.7004073E7,
        "sum_of_squares" : 8.60820897863E11,
        "variance" : 1.4246828534358414E7,
        "std_deviation" : 3774.4971233739752,
        "std_deviation_bounds" : {
            "upper" : 8178.4543824922475,
            "lower" : -6919.534111003653
        }
    }
}
```

在下面的章节中，将借助一些例子来学习更多关于聚合的知识。例如，在第 12 章案例研究——Meetup 中，将使用地理位置相关的聚合，以及词项(term)、范围(range)等其他聚合方式。

2.6 Painless 脚本说明

有时候，我们会使用脚本、更新数据、脚本字段和许多其他案例。在 Elastic Stack 5.x 版本之前，Groovy 是默认的脚本语言，那时甚至没有具体指定脚本。由于这些脚本是远程执行的，所以安全性一直是 Elastic 团队必须解决的问题，这就是设计 Painless 脚本的原因。

谈到性能，Painless 既安全又高效。它的语法与 Groovy 类似，很容易学习和使用。在大多数情况下，无须改变以前写过的脚本，只需添加一个名为 lang 的参数，并将该值指定为 painless 即可。

要在 Painless 脚本中定义变量，只需编写以下内容：

```
def myVar = 'my-value';
```

不需要指定任何类型。在运行时，变量的类型将被检测为适合的类型。Painless 支持所有由 Java 定义的变量。

要定义数组，可以编写以下内容：

```
int[]
```

要定义一个列表，可以编写以下内容：

```
def nums = [1, 2, 3, 4, 5]
   nums[0] // 1st element
   nums[-1] // last element of the list
```

同样，映像也可以被定义：

```
def book = ['name': 'Java', 'pages': 450];
book.name // refers to Java
```

要执行循环操作，可以编写以下内容：

```
for(item:nums) { .. }
```

为了进一步理解如何使用这种脚本，这里使用前面提到的 library 索引。考虑在每本书的结尾加上 5 页作为广告页，图书馆里所有的书都必须更新，并且在页面计数上增加 5 页。要做到这一点，可以用 Document API 中的 update_by_query 实现：

```
curl -XPOST "http://localhost:9200/library/book/_update_by_query" -d'
{
    "script": {
        "inline": "ctx._source.pages +=5",
        "lang": "painless"
    },
    "query": {
        "match_all": {
        }
    }
}
```

在这个简单的例子中，已经使用 ctx. _source 更新了页数，通过使用 match_all 查询为每本书添加了 5 页。

让我们再增加一些复杂性。如果所有的书都有一个出版社字段，类似于以下文档中的书：

```
curl -XPUT "http://localhost:9200/library/book/41/_create?pretty" -d'
```

```
{
  "author" : "Ravi Kumar Gupta",
  "title" : "Test-Driven JavaScript Development",
  "pages" : 240,
  "publication" : "packt"
}'
```

现在,想要增加 5 页,但只为 Packt 出版社出版的书增加页数,就可以用下面的限制条件来实现:

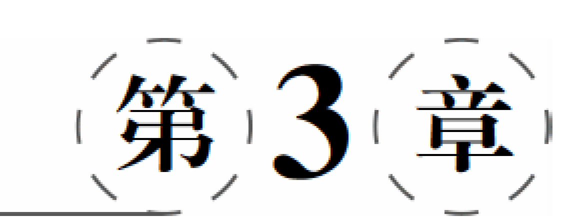
```
curl -XPOST "http://localhost:9200/library/book/_update_by_query" -d'
    {
      "script": {
        "inline": "if(ctx._source.publication ==  \"packt\"))"
        ctx._source.pages +=5",
        "lang": "painless"
      },
      "query": {
        "match_all": {
        }
      }
  }'
```

类似地,脚本中也可以设置循环,以及其他由 Painless 提供的功能。Painless 脚本的完整文档可以在此网址查找:https://www.elastic.co/guide/en/elasticsearch/reference/5.1/modules-scripting-painless.html。

2.7 本章小结

在本章中,了解了 Elasticsearch 的架构及其诞生过程;熟悉了 Elasticsearch API——SearchAPI、IndicesAPI 和 DocumentAPI 等。借助这些 API,学习了如何向 Elasticsearch 中添加文档,如何查询这些文档,以及如何管理索引。在本章的最后,聚合展示了如何有效地搜索文档。在后面的章节中,将用更多的例子来实际运用这些概念。

在下一章中,将了解 Logstash,并学习如何针对复杂数据类型配置 Logstash 以及 Logstash 插件。



探索 Logstash 及其组件

在前面的章节中，已经学习了许多有关 Elasticsearch 的特性、体系架构及它所提供的各种 API 的用法等，也详细地学习了如何使用它提供的 API 及 API 的功能等。在学习完 Elasticsearch 后，本章将开始学习 Elastic Stack 提供的另一个流行的功能组件——Logstash 的用法，学习 Logstash 的需求、Logstash 数据流体系架构及其各种组件的使用方法。

学完本章内容后，就会明白 Logstash 的使用需求、Logstash 数据流以及它提供的各种插件(plugin)的配置方法。

本章将学习如下主要内容：

- Logstash 简介；
- Logstash 插件的体系架构；
- Logstash 配置文件结构；
- 插件的种类；
- 数据输入插件 Input 的用法；
- 数据过滤插件 Filter 的用法[①]；
- 数据输出插件 Output 的用法；
- 编解码插件 Codec 的用法；
- 插件的命令行操作；
- Logstash 的命令行操作；
- 解析日志的 Logstash 配置；
- 监控 API 的用法。

① 译者注：如 grok、date、mutate、ruby、metrics 等，各种数据裁剪和计算都可以在这里完成。

3.1 Logstash 简介

Logstash 起源于 Jordan Sissel 研发的一个智能工程产品。Jordan Sissel 具有操作系统和系统管理员的工作背景，他经常在分析日志的过程中遇到挑战。有一次，他想在秒级时间内处理大量的日志数据流，然而却找不到一款满足这种需求的合适的免费的开源工具或工程来解决这个问题。面对这种挑战，他开始着手为用户构建适合这种需求分析的工具（即后来的 Logstash），以便能在秒级处理这种大量的日志流。在这款软件工具中，他还结合了其他一些强大的功能，以便能从大量日志中获取相关的信息。开发 Logstash 的一个目的是：构建一个数据中心处理系统。利用它，能从多数据源中获取日志，并便于日后日志的处理和数据的聚合分析。

起初，Logstash 是作为一个独立产品出现的。随着 Elasticsearch 的成长，Jordan Sissel 加入到 Elastic 的团队中，并直接负责 Logstash 的研发。从那时起，作为一个能从多种数据源中搜集数据、记录数据、分析日志数据，并和其他组件协同完成存储数据、可视化数据的综合性平台，Logstash 不断成长为 Elastic Stack 的一个核心部件。Logstash 具有一些强大的处理功能，拥有实时搜集数据的能力，这使它成为一款处理日志信息的综合性工具。作为 Elastic Stack 中的一员，Logstash 还是一款能为用户提供端到端解决方案的开源软件。

3.2 为什么需要用 Logstash

Logstash 起初主要是为管理日志而出现的一个工程，但它现在已经扩展到能分析各种类型的数据了，其数据的处理范围涵盖事件数据、时间戳数据、应用程序日志、事务型数据、CSV 数据、输入文件等。数据可以是结构化的、非结构化的，或者是那种难以被转换为某种合适格式的半结构化的数据。为了能管理各种来源于不同系统的不同类型的数据，我们需要一种能处理各种不同类型日志数据的有效的工具，并希望它能以近乎实时的处理方式来分析这些数据。Logstash 能从各种系统中搜集并集中在中心系统中管理，在这里，数据能够被解析并按需求预处理。此外，Logstash 还能从多个系统中获取数据并按照通用的方式存储数据，而这些数据将来可以便于 Elasticsearch 和 Kibana 使用。

Logstash 可通过数据管道（pipeline）来抽取、清洗、传输、载入数据，以便从数据中获取有价值的信息。在这种工作方式下，Logstash 用于对数据的**抽取、转换、载入（ETL）**，而 ETL 是一个在数据仓库和商业智能领域中使用广泛的词汇。Logstash 能从多系统中抽取数据，并执行一些处理或转换日志数据的操作（例如在载入和处理数据后完成过滤数据、移除字段、增加字段、分析数据等操作）。

Logstash经常被比喻为瑞士军刀，因为它能处理任何类型的数据。由于它含有大量的能从各种不同数据源中读取数据的Input插件、转换数据的Filter插件，以及存储、输出数据的Output插件，从而使得Logstash成为一款几乎能处理所有数据的常用工具。

先让我们看看Logstash的一些关键特点。

3.3 Logstash的特点

在Logstash诸多的特点中，下面列出了使Logstash有别于其他日志管理工具的关键特点。

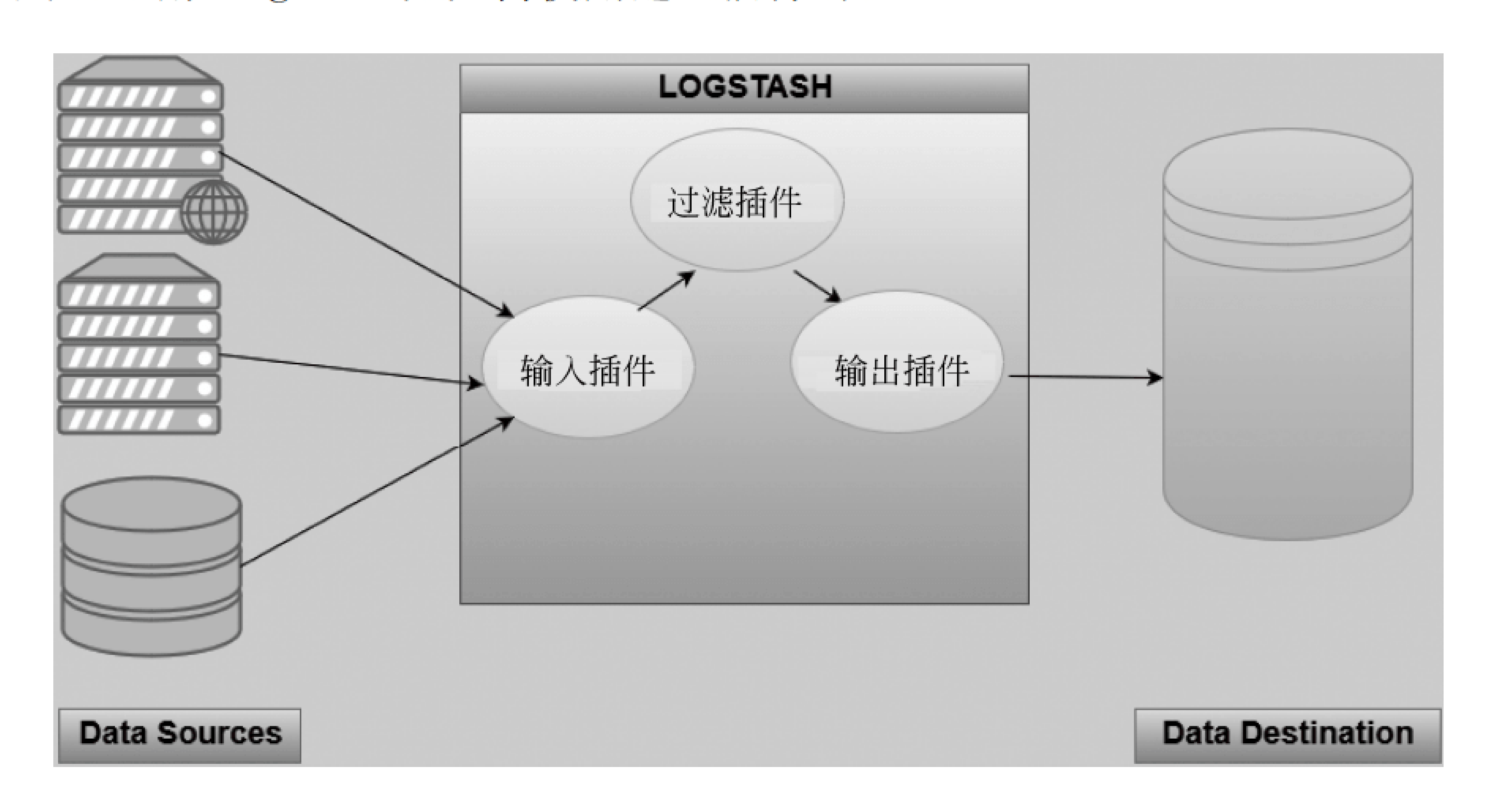
- **开源**：作为一款开源工具，Logstash是可以完全免费使用的，其源代码可以从Github上免费得到。
- **与Elasticsearch、Beats、Kibana无缝连接**：Logstash是一个基于管道式（pipeline）处理数据的强有力的工具。这个数据处理管道能依靠紧密耦合在一起的Beats（用于从多系统获取数据）、Elasticsearch（完成存储数据、实时检索）、Kibana（完成可视化数据的操作）等，完成相应的数据处理操作。
- **可扩展性**：Logstash提供多种输入Input、过滤Filter、输出Output插件，以便处理各种不同类型的日志数据。它在生成和开发Logstash的插件（Input、Filter、Output）上提供一定的灵活性，使得基于Logstash的研发变得更加友好。
- **互操作性**：Logstash在使用其他各种插件和工具方面提供互操作性。Logstash能从各种数据来源（或工具）获取数据，并将数据输出到不同的目的地（工具）。这使Logstash能成为复杂的数据处理工作流中的一部分，并使数据处理变得更加容易。
- **可插入式的数据管道处理体系架构**：Logstash包含超过200种由Elastic和社区开发的可用的数据插件。Logstash被设计成为一种“泛型”体系结构来处理数据，可以在其中增添输入、过滤、输出的各种插件，它使用一种配置文件的格式来使用各种插件。

为便于使用这些不同的数据插件，Logstash提供了相应的配置文件。这是其体系架构的关键和核心，以确保其拥有足够的灵活性。为更好地理解这一点，先来看看如下的例子。

假定这里有一个用来存储数据的新工具或新软件，Logstash目前不支持它，无法与存储在其中的数据进行交互。如果有了这种可插入式的体系架构，就可以很容易地通过写入一些代码和使用一系列参数来开发一个数据插件，并将其直接使用在Logstash中。你不必深究Logstash源码，就能很容易地开发一个插件。数据插件既能扩展Logstash的应用范围，又能增加对相应数据的处理能力。

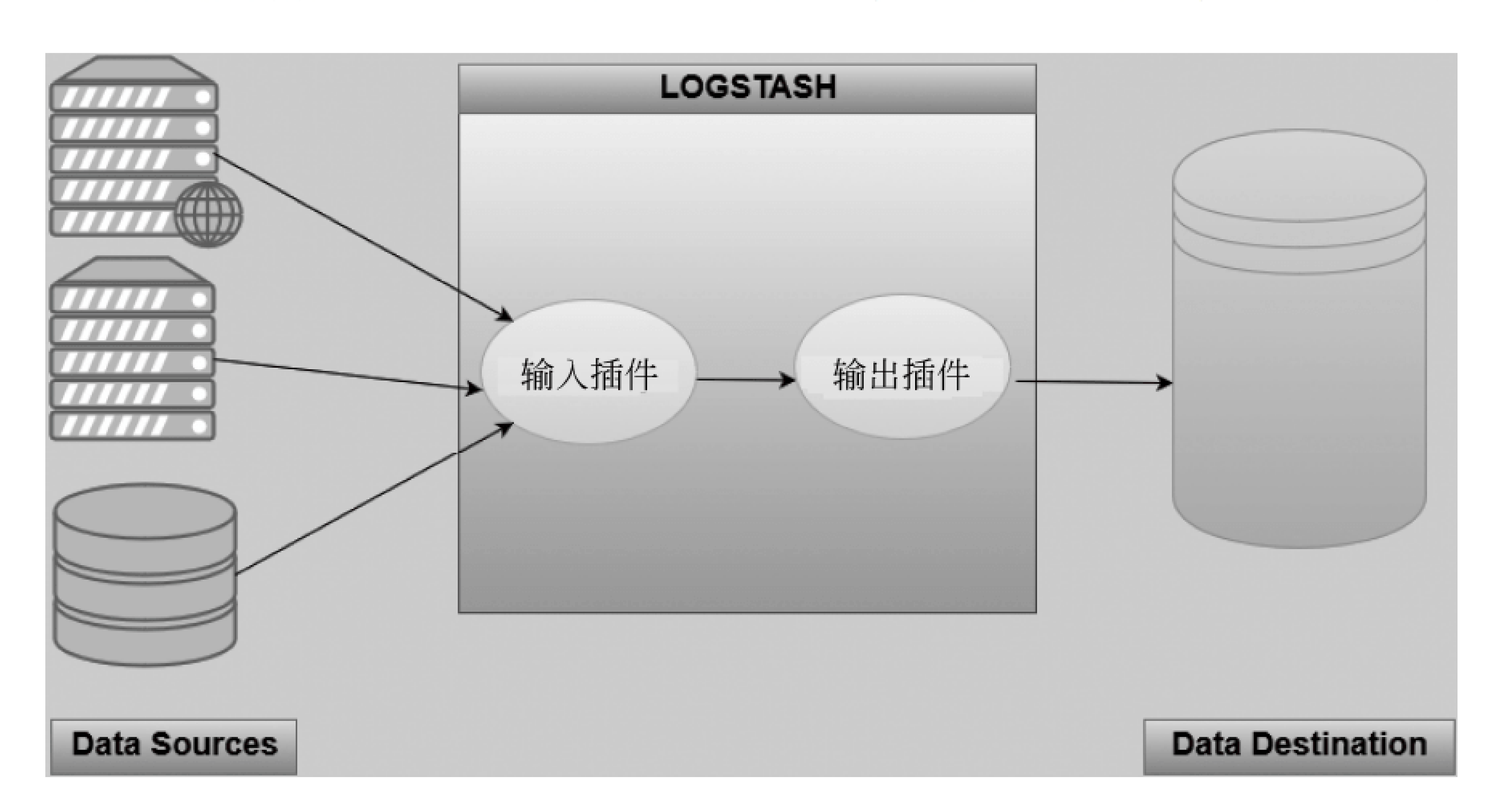
3.4 Logstash 插件的体系架构

基于 Logstash 的数据处理管道包括输入（Input）、过滤（Filter）、输出（Output）插件。先通过下图，了解 Logstash 是如何使用这些插件的。



在上图的体系架构中，能看到这里有便于数据搜集的多种数据源，它们在 Logstash 中是作为 **Input 插件**的；在得到输入的数据后，使用 **Filter 插件**来转换这些数据；处理完数据后，借助 **Output 插件**，把数据存储或写到相应的目的地中。

Logstash 使用配置文件来设定（存储）相应的输入数据、过滤数据、输出数据的插件。**数据输入 Input 插件**和**输出 Output 插件**是在配置文件中强制使用的数据处理插件，而**过滤 Filter 插件**则是可选的。如果已经准备好输入数据，不要求转换和更改这些输入数据，并且把数据直接存储到某个目的地，则在这种情况下，Filter 插件不是必需的。其系统架构可简单地由下图表示。



既然已经明白了 Logstash 的系统结构，下面就来看看其配置文件。在配置文件中，可以定义各种输入数据、过滤数据、输出数据的插件。

3.5　Logstash 配置文件的结构

一个 Logstash 配置文件包括 input、filter、output 这三部分，每部分用于设定想要使用的相应的数据插件。先看下面的例子：

```
input {

      }
filter {

       }
output  {

        }
```

其中每部分均包含用于配置单个或多个插件的配置信息。为了能配置插件，需要在相应部分中提供插件的名称，并提供其设置信息或参数（由“键＝＞值”对的方式表示）。**在每部分中，如果定义了多个插件，则它们的执行顺序就是在配置文件中出现的顺序**。每一个插件都有自己相应的设定参数，这些设定参数需要和相应的插件名称对应起来。

> 这里的“＝＞”符号表示一个指派操作，它用于将一个值 value 复制给在配置文件中对应的键 key。

在深入了解每章可用的各种插件之前，先要明白各种“值”的“类型”，以及在配置文件中如何使用条件陈述。这些“类型”可以在配置文件中通过“值”来赋值。

3.5.1　值类型

每一个 Logstash 的插件都包括一系列设定值。对于一些字段来说，有些设定值是必需的。对每一个设定值，都要定义一系列 Logstash 能够支持的不同取值。下面介绍可用的值类型。

3.5.1.1　数组类型

数组（Array）具有一个或多个数值。

对于单值情况，其句法如下：

```
Key=>"value"
```

对于多值情况，其句法如下：

```
Key=>["value1","value2", "value3"]
```

如果在一个数组中多次设定一个值，相应的值就会追加到数组中，如下所示：

```
key =>"value"
key =>"value1"
key =>["value2", "value3", "value4"]
```

这个 key 会存储 5 个值(即 value、value1、value2、value3 以及 value4)。

3.5.1.2 布尔类型

布尔(Boolean)类型用于设定非真(true)即假(false)的值。

例如：

```
Key =>true
Key1 =>false
```

> ℹ布尔类型的值(即 true 或 false)禁止放在引号中使用。

3.5.1.3 字节类型

字节(Byte)是字符串类型的数据(在双引号中引用)，用来表示字节的集合。它可使用**国际单位制(SI Units)**(kB、MB 或 GB)以及二进制(KiB、MiB、GiB)计算字节数(国际单位制 SI Units 用 1000 表示千，而二进制用 1024 表示千，即：1kB＝1000 字节，1KiB＝1024 字节)。Byte 可用来定义值。它是大小写敏感的，接受 key 值和 unit 间的空格。

例如：

```
size =>"2467KiB"
size =>"9872MiB"
key =>"452GB"
```

> ℹ如果未指定单元，则值表示字节的数量。

```
Key =>"1234"
```

3.5.1.4 Codec

Codec 不是一种值类型，但它常常用来表示数据。它用于按照需求解码那些从数据输

入插件 Input 中读取的数据,并在输出到数据输出插件 Output 前进行编码。使用它,就不必再使用那些设定数据格式的特定的 Filter 插件了。

```
codec =>"plain"
```

3.5.1.5 注释

这不是一种值类型,而是在配置文件中定义说明文字的注释。其语法类似于 Perl、Python、Ruby 等。它由#来标识,可以出现在任意语句行中。

```
Key =>"value"   #这是一种字符串类型
#希望你现在正在学习中哦
```

3.5.1.6 哈希类型

哈希(Hash)类型是一组键值对,其中键和值都是用双引号表示的。多个键值对之间要用空格分隔,不能用逗号。

```
match =>{
    "field1" =>"value1"
    "field2" =>"value2"
    …
}
```

3.5.1.7 数值类型

数值(Number)类型的数据包括合法的整型或浮点型的数据。

```
number =>44
amount =>1.28
```

3.5.1.8 字符串类型

字符串(String)类型的值可用单引号或双引号标识。如果字符串本身在定义时就含有单引号或双引号的内容,则需要用反斜杠来转义。

```
name =>"yuvraj"
escape =>"value\"ue"
single =>'Hello It\'s nice you are reading it'
```

3.5.2 条件判断的用法

条件判断语句用于检查语句能被执行的条件是否满足。在配置文件中,能检查插件执

行的条件，其处理方式和其他编程语言相似，支持 if、else 和 else if 语句。

配置文件中的条件语句块结构如下：

```
if EXPRESSION  {
    ...
} else if EXPRESSION  {
    ...
}else  {
    ...
}
```

表达式中包括一些操作符(如比较操作符、布尔操作符、一元操作符等)。比较操作符可分为等值操作符、正则表达操作符、集合操作符等。

- **比较操作符**：==、!=、<、>、<=和>=。
- **正则表达操作符**：=~和!~。
- **集合包含操作符**：in 或 not in。
- **布尔操作符**：and[①]、or、not 或 xor。
- **一元操作符**：!。

在本章后续内容中，将会看到如何使用这些操作符的例子。

如上所述，Logstash 配置文件可分为多个部分。下面就来看看在这些部分中各种可用的插件。

3.6 插件种类

在下面的内容中，我们来看看一些插件的用法。

3.6.1 数据输入插件 Input

在 Logstash 中，各种可用的输入插件 Input 如下图所示。

详情可查看该页面的内容：https://www.elastic.co/guide/en/logstash/5.1/input-plugins.html。

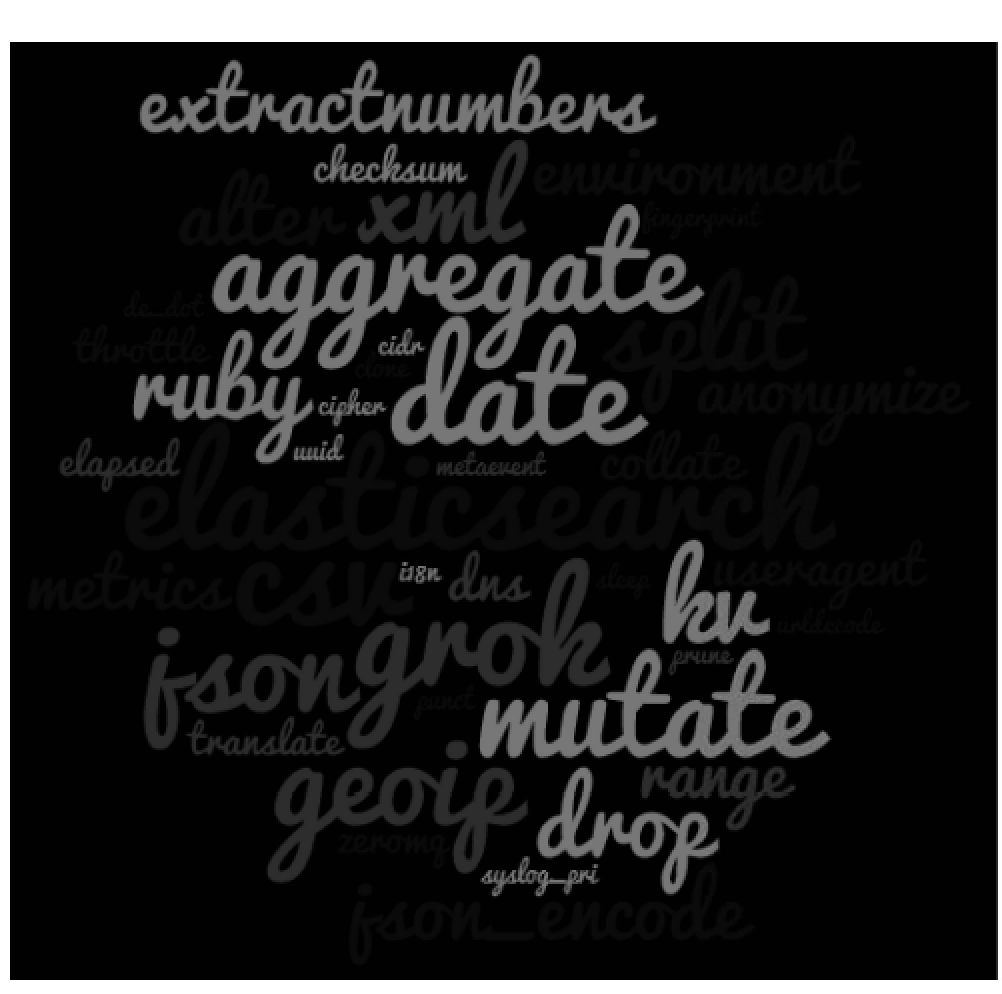
3.6.2 数据过滤插件 Filter

Logstash 中可用的数据过滤插件 Filter 如下图所示。

① 译者注：此处原文为 nand，笔者认为此操作符应为 and。



Input插件



Filter插件

详情可查看该页面的内容：https://www.elastic.co/guide/en/logstash/5.1/filter-plugins.html。

3.6.3　数据输出插件 Output

Logstash 中可用的各种数据输出插件 Output 如下图所示。

详情可查看此页面的内容：https://www.elastic.co/guide/en/logstash/5.1/output-plugins.html。

上面已经列出了在配置文件中各种不同的可用插件(即 Input、Filter、Output 各种插件)。下面还有一些位于 Input 和 Output 中的不同类型的插件。

3.6.4　编解码插件 Codec

作为数据输入和输出块中的一部分，Codec 是可能出现在其中的一部分内容。它用来



Output插件

表示一个事件中的数据，也被称为数据流过滤(stream-filter)插件，因为它提供了一种进入(或离开)Logstash 的编码(或解码)的数据流处理方式。

在 Logstash 中各种可用的 Codec 插件如下图所示。



Codec插件

详情可查看该页面的内容：https://www.elastic.co/guide/en/logstash/5.1/codec-plugins.html。

3.7 学习数据输入插件 Input

数据输入插件 Input 用于从单一或多源数据源中获取数据，并将其输送到 Logstash 中。在 Logstash 的配置文件中，它往往是作为第一部分出现的。本节后续部分将会介绍一些输入插件的使用。

3.7.1 stdin

stdin 是一个相对简单的插件，用于从标准输入设备中读取数据。它可以读取从控制台输入的数据并将数据传送到 Logstash 中。它常用于验证 Logstash 是否正确安装，以及是否能读取 Logstash 中的信息。

在配置文件中，有关 stdin 的基本用法如下：

```
stdin {

}
```

在这个插件中没有必须要设置的参数项。如果使用上述设置(即未提供其他额外的可选参数项)，则在控制台输入的信息将作为 Logstash 的输入。

虽然没有必选项，但也可以使用如下一些可选项。部分可选项的设置如下所示：

- add_field：向输入数据中增加字段。
- codec：编解码输入的数据，并说明作为 input 数据的格式。可用的取值是 Codec 插件所支持的取值。
- enable_metric：在每个插件汇报自己的性能指标测度时，接收(也可设置为关闭接收)各项指标的值。
- id：为相应插件提供唯一的标识符，可用于跟踪相应插件的运行轨迹信息，也可用于对该插件的调试。
- tags：为输入数据增加可用于信息处理的标记。对于相似的事件处理，可为从给定输入源得到的数据增加标记。它常用于条件判断中，可对每一个标记执行各种不同的操作设置。
- type：为输入数据增加 type 字段。从多个不同的数据源输入异构数据时，type 字段是非常有用的，它能从众多数据源中区分各自的数据。type 常用于过滤 filtering 中，使用 type 可以从不同数据源中区分不同的处理逻辑[①]。

值类型及其默认值如下表所示。

设　置	值　类　型	默　认　值
add_field	哈希	{}
codec	codec	“line”

① 译者注：在输入数据中，可以用 type 来编辑事件或数据类型；tags 则是在数据处理过程中由具体的插件来添加或删除的。

续表

设　置	值 类 型	默 认 值
enable_metric	布尔	true
id	字符串	无
tags	数组	无
type	字符串	无

下面是一个配置文件示例：

```
input {
        stdin {
                add_field =>{"current_time" =>"{@timestamp}"}
                codec =>"json"
                tags =>["stdin-input"]
                type =>"stdin"
              }
      }
```

在上面的配置中，提到了基于 JSON 的编解码。输入是基于 JSON 格式的。进入 JSON 的内容将会被解析为一个个的键值对，并增添一个 current_time 字段，其值来源于元数据字段的 timestamp 字段。如果输入任何非 JSON 格式的数据，将会在 stdin_input 的 tags 中返回_jsonparsefailure 的信息。

3.7.2 file

插件 file 是最常见的从文件中读取信息的插件。它表示从一个文件或一组文件中读取的数据流 stream(类似于使用尾标识-F)①，但其功能不仅限于此。作为一个功能强大的插件，它能完成很多功能，如跟踪文件中的任何变化，读取文件的最后位置，用数据更新该文件，检测文件的循环使用情况，提供从头到尾读取文件的操作等。

它保存从 sincedb 指定的位置读取的数据②。默认情况下，它是存放在 $HOME 目录下的，但其位置可以由 sincedb_path 设置来改变。跟踪文件的频度可以由 sincedb_write_

① 译者注：Logstash 使用 start_position 来确定从什么位置开始读取文件数据，默认是结束位置，进程会以类似 tail-F 的形式运行；如果把 start_position 属性改为 beginning，则进程就从头开始读取 ongoing，但读到最后一行不会终止，而是继续变成 tail_F。

② 译者注：sincedb 是一个文件，用来跟踪被监听日志文件的当前读取位置。

interval 设置来改变[①]。

对 file 的基本配置如下所示：

```
file{
    path =>…
}
```

在这个插件中，只有 path 设置是必须要给出的（其他都是可选的）。

3.7.3　path

这个插件定义拟读取的目录或文件的存放位置。可以提供目录名、指定的文件名、待查找的文件名的模式，还可以指定单个或多个拟查找的模式/文件位置。

> 对插件 path 的定义，必须使用绝对而非相对路径。

其他可选配置的设定如下：

- add_field：对输入的数据增加字段。
- close_older：关闭最后修改时间比设定时间还要久的输入文件。它可以释放文件的 I/O 操作并反复检测针对文件的改动。如果一个正在被追加数据的文件未能及时获取数据，并且操作用时已经超出设定的时间限制，那么这个文件将会被关闭，以便其他文件能被打开。之后，每当对文件更新或修改时，被关闭的文件才会重新进入队列排队并重新打开。
- codec：编解码输入数据，并解释输入数据的格式。
- delimiter：识别文件中不同行的分隔符。
- discover_interval：定义搜索间隔值。这些设置值用于决定每隔多久去检查一次被监听的 path 下是否有新文件。
- enable_metric：从每个插件中得到测度值，并用于对插件的报告。
- exclude：排除掉不作为输入（即：监听）的文件或文件模式。
- id：为插件提供一个唯一的标识符，可用于跟踪插件的信息，也可用于对插件的调试。
- ignore_older：忽略读取自从某个时间（可设定）以来未被修改的文件。即使文件被忽略，也可被读取。
- max_open_files：定义同时打开文件的最大数量。如果需要读取超过这个阈值的更多的文件，可以使用 close_older 参数来关闭那些最近未被修改过的文件。如果这

① 译者注：如果不想用默认的 $HOME/.sincedb，可以通过 sincedb_path 配置定义 sincedb 文件到其他位置；sincedb_write_interval 定义 Logstash 每隔多长时间写 yieldsincedb 文件。

个属性值设置得非常高,可能会导致操作系统性能方面的问题。

- sincedb_path:定义写 sincedb 文件位置,它用来记录日志文件的轨迹。
- sincedb_write_interval:确定写 sincedb 文件的时间间隔,包括用于跟踪多日志文件的当前读写位置。
- start_position:**从头 beginning 至尾 end**。定义从哪里开始读取文件中的信息。当文件被初次读取且在 sincedb 文件中未存放相应信息时,才使用这个参数。如果文件包括旧的数据,就可以使用这个参数来确定从文件的什么位置(而非默认的位置)开始读取文件中的信息。
- stat_interval:检查文件是否被修改过。增大这个间隔值,将导致系统发出更多更频繁的调用,但如果需要跟踪日志文件,这个数值应该调得大些。
- tags:为输入数据增加可用作日后处理的标记。它多和条件判断一起使用,每个 tag 部分执行一种操作。多 tag 的使用,可有助于执行各种不同的操作。
- type:为输入数据增加 type 字段标记。从多个不同的数据源中读取数据时,该功能是很有用的,因为使用 type 可以从众多不同的数据源中区分不同数据。它主要用于过滤,可以使用 type 字段为不同的数据源增加不同的处理逻辑。

上述提到的值类型以及它们的默认值如下表所示。

设置字段	值类型	默认值
add_field	哈希	{}
close_older	数值	3600
codec	codec	“plain”
delimiter	字符串	“n”
discover_interval	数值	15
enable_metric	布尔	true
exclude	数组	无
id	字符串	无
ignore_older	数值	无
max_open_files	数值	无
path	数组	无
sincedb_path	字符串	$ HOME/. sincedb *
sincedb_write_interval	数值	15
start_position	字符串	“end”

续表

设置字段	值类型	默认值
stat_interval	数值	1
tags	数组	无
type	字符串	无

配置文件示例：

```
input {
        file {
              path => ["/var/log/elasticsearch/ * ","/var/messages/ * .log"]
              add_field => ["[location]", "%{longitude}" ]
              add_field => ["[location]", "%{latitude}" ]
              exclude => [" * .txt"]
              start_position => "beginning"
              tags => ["file-input"]
              type => "filelogs"
             }
       }
```

在上述的配置文件中，我们已经看到一组路径 path，文件可被从这里读取。从 elasticsearch 文件夹中读取文件，这些文件在 message 文件夹中，以 LOG 作为文件的扩展名。假设文件都是有 latitude 和 longitude 位置信息的，增加一个 location 字段，其中包括文件的 latitude 和 longitude 位置信息（其格式是 Kibana 中读取 geoip 数据的格式）。我们排除了配置文件中所有的文本文件。配置文件告诉 Logstash 从开头读取文件，文件中含有想要读取的信息。

上述配置文件可分割为两个不同的文件，其 path 参数的值根据设定的不同的 tags 和不同的类型而有所不同，代码片段如下所示。

```
input {
    file {
             path => "/var/log/elasticsearch/ * "
             tags => ["elasticsearch"]
             type => "elasticsearch"
         }
    file {
             path => "/var/messages/ * .log"
             tags => ["messages"]
             type => "message"
```

```
        }
}
```

3.7.4 udp

插件 udp 用于基于 UDP 协议从网络中来读取数据。它会列出一组端口 port，从这些端口可以读取基于 UDP 协议的事件数据。

基于 udp 的基础配置文件如下所示：

```
udp {
        port =>...
}
```

在这个插件中，只有端口号 port 的值是必选项(其他均为可选项)：

- port：Logstash 从这个端口来侦听读取的事件或信息。

> 如果使用小于 1024 的端口号，则需要提供使用 root 的特权。

除上述的 port 属性外，其他可选的设置信息如下：

- add_field：向输入数据中增加字段。
- buffer_size：定义从网络中读取数据的最大的数据包大小。
- codec：解码输入的数据，并解释输入数据的格式。
- enable_metric：从每个插件中得到测度值，并用于对插件的报告。
- host：设定 Logstash 将要侦听的主机名称。
- id：为插件提供一个唯一的标识。该 id 号可用于跟踪插件的信息，并可用于对其进行调试。
- queue_size：设定可驻留在内存中的未处理的数据包的最大数目。如果待处理的数据包的数目大于这个设定的 queue_size 值，则相应的数据将会被丢弃。
- receive_buffer_bytes：用于定义接收的缓存大小(字节数)。如果不设定该值，则使用操作系统默认值。
- tags：为读入的数据增加可用于后续处理的标记。该标记多用于条件语句，可用于在每一个 tags 中执行相应的不同操作。
- type：为输入数据增加 type 字段标记。从多个不同的数据源中读取数据时，该功能是很有用的。使用 type，可以从众多不同的数据源中区分不同的数据。它主要用于过滤，可以使用 type 字段，为不同的数据源增加不同的处理逻辑。
- workers：定义处理数据包的线程数量。

上述提到的值类型以及它们的默认值如下表所示。

设　　置	值　类　型	默　认　值
add_field	哈希	{}
buffer_size	数值	65536
codec	codec	"plain"
enable_metric	布尔	true
host	字符串	"0.0.0.0"
id	字符串	无
port	数值	无
queue_size	数值	2000
receive_buffer_bytes	数值	无
tags	数值	无
type	数组	无
workers	字符串	2

配置文件示例：

```
input{
    udp {
        host =>"192.168.0.6"
        port =>5000
        workers =>4
    }
}
```

在上面的配置示例中，我们已经注意到使用了 host 和 port，Logstash 会从这里读取事件和数据。我们还设定 workers 阈值是 4，表示有 4 个线程将用于并行处理网络数据包。

3.8　学习数据过滤插件 Filter

过滤插件 Filter 执行对数据的转换。如果需要对获取的输入数据执行某种转换操作，使用 Filter 插件能(在将数据输出前)完成对数据的相关操作[①]。它在数据的输入和输出之间扮演者中间人的角色。该插件需要在 Logstash 的配置文件中进行设置。

首先让我们看看一些过滤插件。

① 译者注：它们扩展了进入到过滤器的原始数据，增加了可能复杂的逻辑处理(如可添加新的事件 event)。

3.8.1 grok

插件 grok 是在 Logstash 中最常使用的一种将非结构化数据转换为结构化数据的功能强大的插件。即使是结构化的数据，也可以使用这种模式完成数据序列化操作。由于 grok 模式的强大且自然的功能，Logstash 也被称为瑞士军刀。grok 可以完成解析且结构化数据的任务。只要是可读取的日志文件，它都可解析。它将文本模式结合到数据结构化中，将相应的日志文件匹配、组织并结合到相应字段中。Logstash 提供了超过 120 种便于使用的 grok 模式，也可以生成适合自己需求的匹配 grok 的新模式。

> grok 模式的资源可以在这里下载：https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns。随着 Logstash 的安装，它们也安装到相应的文件夹中，可以通过搜索 grok 模式文件找到它们。

grok 模式的基本语法如下：

```
%{SYNTAX:SEMANTIC}
```

这里，SYNTAX 是用于匹配数据的模式(或正则表达式)名称，SEMANTIC 是标识符或字段名称(它提供匹配的模式或正则表达式，如在语法中定义的那样)。

有了 grok，就可以使用正则表达式或者生成自己个性化的模式。使用的正则表达式库是 **Oniguruma**，其正则语法可下载：https://github.com/kkos/onigurama/blob/master/doc/RE。可以生成个性化的模式文件，其中包括用于匹配相关数据的自定义模式。grok 从左到右依次匹配数据，并且一个一个地匹配相应模式。

下面针对一些日志文件，来看看如何使用 grok 模式来匹配它们。

假如有如下一条简单的日志行：

```
Log line: Jun 19 02:11:30 This is sample log.
```

使用现有的 grok 模式生成一条匹配上述日志信息行的 grok 模式。

```
%{CISCOTIMESTAMP:timestamp} %{GREEDYDATA:log}
```

为便于理解，让我们来看一看规定的模式。

```
CISCOTIMESTAMP %{MONTH} +%{MONTHDAY}(?: %{YEAR})? %{TIME}
GREEDYDATA .*
```

看起来不好理解。怎么知道哪个模式能匹配我们的数据？

我也同意这个说法，毕竟不好理解哪一种模式可用，哪一种模式能匹配我们的数据。但不必担心，你的朋友们将会帮助你发现适合你的模式，以便能针对每一个需求修改对应的模

式。可以访问下面两个网站获取帮助，它们能帮助你构建匹配你的数据的模式：

- http://grokdebug.herokuapp.com
- http://grokconstructor.appspot.com/

有关 grok 的基本配置信息如下：

```
grok {

}
```

使用这个插件时没有必选项。其他可选项的设置值如下：

- add_field：为输入的数据增添字段。
- add_tag：为输入的数据增添 tag。这些 tag 可以是输入数据中基于 key 值的静态或动态的 tag。
- break_on_match：如果设置为 true，grok 的第一个成功匹配将结束 filter 操作。如果它成功匹配模式，将会完成过滤操作。若想完成所有的匹配，它应该设为 false。
- keep_empty_captures：如果设为 true，它将被用于保存空的事件字段。
- match：匹配有值的字段。该值用于单模式或多模式(多模式可以以数组方式提供)。
- named_captures_only：如果该值设置为 true，则用于使用 grok 模式定义的字段。
- overwrite：覆盖字段中的值。它将会复写已经存在的字段值。
- patterns_dir：指向目录。可以定义个性化的模式，可以定义单个目录或多个目录。
- patterns_files_glob：如果设置为 patterns_dir，则用来选择目录中的所有文件。
- periodic_flush：在规定的时间间隔，周期性地调用 flush 方法(定期调用清理缓存的一种方法)。
- remove_field：从输入数据中移除字段。
- remove_tag：从输入数据中移除 tag。
- tag_on_failure：如果事件无法匹配 grok 模式，或者没有成功匹配特定值的模式，则返回 tag 出错信息。
- tag_on_timeout：如果 grok 正则表达式超时，则产生新的 tag。
- timeout_millis：如果超过设定时间，则中断正则表达式。如果有多个针对 grok 的模式，则对每一条模式均有效。它是毫秒级的。

上述设置的值类型以及它们的默认值如下表所示。

设　置	值类型	默认值
add_field	哈希	{ }
add_tag	数组	[ ]

续表

设　　置	值　类　型	默　认　值
break_on_match	布尔	true
keep_empty_captures	布尔	false
match	哈希	{ }
name_captures_only	布尔	true
overwrite	数组	[ ]
patterns_dir	数组	[ ]
patterns_files_glob	字符串	"*"
periodic_flush	布尔	false
remove_field	数组	[ ]
remove_tag	数组	[ ]
tag_on_failure	数组	["_grokparsefailure"]
tag_on_timeout	字符串	"_groktimeout"
timeout_millis	数值	2000

配置文件示例：

```
filter {
    grok {
        add_field =>{"current_time" =>"%{@timestamp}" }
        match =>{ "message" =>"%{CISCOTIMESTAMP:timestamp}%{HOST:host} %
        {WORD:program}: \ [%{NUMBER:duration}]%{GREEDYDATA:log}" }
        remove_field =>["host "]
        remove_tag =>["grok","test"]
    }
}
```

在上面的配置中，我们利用定义的模式来匹配数据。如果模式未被匹配，则会增加一个_grokparsefailure 的 tag。它也将会从信息中移除 host 字段。

3.8.2　mutate

mutate 过滤插件用于执行各种各样的对字段的重构和处理（例如对字段的重命名，连接字段，将字段转为大写或小写，拆分字符串或者转换字段的数据类型等）。

有关 mutate 的基本配置操作如下所示：

```
mute {

}
```

使用这个插件时没有必选项。其他可选项的设置值如下：

- add_field：用于向输入数据中增加字段。
- add_tag：用于为输入的数据增添 tag。这些 tag 可以是输入数据中基于 key 值的静态或动态的 tag。
- convert：用于转换字段的数据类型。可以将字段值转换为整型、字符串、布尔型、日期型或浮点型等。

> 所有存储在 Elasticsearch 中的字段数据类型都是字符串型的。如果需要转换相应字段的数据类型，可使用 convert 来完成转换。

- gsub：用正则表达式完成搜索和替换字段中值的功能。类似于 UNIX 系统中的 sed 命令。其输入有三个参数（字段名、搜索模式、替换）。它仅对字符串类型字段有效。

> 在搜索模式中不要忘记反斜杠。

- join：用定义的字符来连接数组中的值，只用于数据类型是数组的字段。
- lowercase：将西文字符值转换为小写。
- merge：对数组或哈希数据类型的两个字段的连接操作。字符串类型的字段将会转换为一个数组，以便能使用数组将其连接起来[①]。
- periodic_flush：在规定的时间间隔内，周期性地调用 flush 方法。
- remove_field：从输入数据中移除字段。
- remove_tag：从输入数据中移除 tag。
- rename：为字段改名。
- replace：替换字段中的值。
- split：用一个分隔符来拆分某个字段，使其成为一个数组。它只在字符串数据类型的字段中起作用。
- strip：删除字段值中的空格。
- update：更新某个字段的值。如果该字段不存在，则 update 不起作用。

---

① 译者注：如果拟操作字段是字符串，则会自动先转换为一个单元素的数组，再完成合并；即使目的字段不是数组，也会被强制转换。

- uppercase：将西文字符值转换为大写。

上述设置的值类型以及它们的默认值如下表所示。

设　　置	值　类　型	默　认　值
add_field	哈希	{ }
add_tag	数组	[ ]
convert	哈希	无
gsub	数组	无
join	哈希	无
lowercase	数组	无
merge	哈希	无
periodic_flush	布尔	false
remove_field	数组	[ ]
remove_tag	数组	[ ]
rename	哈希	无
replace	哈希	无
split	哈希	无
strip	数组	无
update	哈希	无
uppercase	数组	无

配置文件示例：

```
filter {
    mutate {
        convert => {"field1" => "float" }
        gsub => ["field2","!","+"]
        lowercase => ["field2"]
        rename => {"field1" => "newfield"}
        strip => ["field1", "field2"]
        update => {"field2" => "It's easy to update"}
    }
}
```

在上面的配置文件示例中，给出了 mutate 的应用示例。

3.8.3 csv

插件 csv 用于对接收到的输入 CSV 类型的数据执行各种操作，解析由逗号分隔开的 CSV 类型的数据。虽然它是 CSV 类型的数据的过滤插件，但它在解析有任何分隔符分开的数据方面，也有不俗的表现。

基于 csv 的基础配置如下所示：

```
csv {

}
```

在这个插件中，没有哪种配置是必选的。可选的设置如下所示：

- add_field：为输入的数据增加字段。
- add_tag：为输入的数据增添 tag。这些 tag 可以是输入数据中基于 key 值的静态或动态的 tag。
- autogenerate_column_names：如设置值为 true，则自动为字段生成名字。如果提供一个 header，它也可被用作列名。
- columns：为在文件中出现的数据列定义名称。如果未指定数据列的名称，则使用默认的列名，如 column1、column2 等。如果有大量的列，则列名自动编号。它可用于当文件没有 header 时对相应列名的定义。
- convert：对字段数据类型的转换。可以将字段值转换为整型、字符串型、布尔型、日期型或浮点型。
- periodic_flush：在规定的时间间隔内，周期性地调用 flush 方法。
- quote_char：指定字符串，引用 CSV 字段中的值。
- remove_field：从读取的数据中去掉某些特定的字段。
- remove_tag：从读取的数据中去掉某些特定的 tag 标记。
- separator：定义列的分隔符。
- skip_empty_columns：定义是否跳过空白的列。
- source：扩展 source 字段的值。
- target：设定存储数据的目标字段。

上述设置的值类型以及它们的默认值如下表所示。

设　置	值 类 型	默 认 值
add_field	哈希	{ }
add_tag	数组	[ ]

续表

设　　置	值　类　型	默　认　值
autogenerate_column_names	布尔	true
columns	数组	[ ]
convert	哈希	{ }
periodic_flush	布尔	false
quote_char	字符串	“"”
remove_field	数组	[ ]
remove_tag	数组	[ ]
separator	字符串	“,”
skip_empty_columns	布尔	false
source	字符串	“message”
target	字符串	无

配置文件示例：

```
filter {
    csv {
        columns => ["id","name","money"]
        convert => {"id" => "integer", "money" => "float"}
        quote_char => "#"
        separator => "	"
    }
}
```

在上面的配置文件示例中，能看到前面提到过的 columns。我们转换了字段的数据类型，改变了引用字符的值，还指定了分隔符用 tab 分隔数据列。

3.9 学习数据输出插件 Output

数据输出插件 Output 用于将数据发送到目的地。在 Logstash 的配置文件中，它是最后一部分。下面介绍常用的输出插件。

3.9.1 stdout

这是一个相对简单的插件，用于把数据输出到 shell 程序的标准输出端，可用于调试使

用插件的配置文件。它常用于验证 Logstash 是否正确解析输入的数据与是否正确使用过滤器(如果有的话)。

stdout 的基础配置文件如下所示：

```
stdout {

}
```

在这个插件中，没有必须要指定的设置值。部分可选项如下所示：

- codec：对即将发送到输出端的数据进行编、解码。能(将数据)使用 Codec 以 JSON 格式来展示 JSON 格式的输出数据，或者(将数据)使用 Codec 以 rubydebug 格式来展示输出使用了 Ruby Awesome Print 库的数据。
- workers：定义将要处理输出数据包的线程数量。

上述设置的值类型及其对应的默认值如下表所示。

设　　置	值　类　型	默　认　值
codec	codec	“line”
workers	数值	1

配置文件示例如下：

```
output {
    stdout {
        codec => rubydebug
        workers => 2
        }
}
```

在前面的设置中，我们已经注意到 rubydebug 格式的 Codec，它将输出结果打印到标准输出设备 shell 上。

3.9.2　file

文件插件 file 用于向文件中输出信息。它能生成存储输出信息的文件，该文件日后也可以使用。默认情况下，它以 JSON 格式向文件中输出信息，每行是一个事件(event)，并可通过编、解码插件 Codec 完成对某种数据格式的转换。

file 的基本配置信息如下所示：

```
file {
          path => ...
```

```
}
```

在这个插件中，只有 path 属性是必须要设置的(其他均为可选项)。

- path：指定文件将要输出的路径。它可以是某个目录的位置，也可以是某个文件的名字(文件将被写入到这里)。可以直接提供文件的名字，或者使用字段名来创建该文件。

其他可选项如下：

- codec：在数据输出前对数据的编、解码操作。
- create_if_deleted：产生一个文件(即使文件可能已被删除)。针对输入数据的解析完成后，信息将会存到该输出文件中(如果该文件已经被删除，因为设置过这个属性，系统就会重新生成该文件)。
- dir_mode：定义使用目录的模式。
- file_mode：定义使用文件的模式。
- filename_failure：如果提供的路径不正确，则所有输出信息将会写入这个文件中。在配置文件针对该属性的字段中，完成对相关属性的设定工作。
- flush_interval：确定写入文件的设定时间间隔(秒级)。
- gzip：把信息写入或存储到文件前，对相关信息以 gzip 格式输出。
- workers：定义将要处理输出数据包的线程数量。

上述设置的值类型以及它们的默认值如下表所示。

设　置	值 类 型	默 认 值
codec	codec	“json_lines”
create_if_deleted	布尔	true
dir_mode	数值	−1
file_mode	数值	−1
filename_failure	字符串	“_filepath_failures”
flush_interval	数值	2
gzip	布尔	false
path	字符串	无
workers	数值	1

配置文件示例如下：

```
output {
          file {
```

```
            create_if_deleted =>true
            file_mode =>777
            filename_failure =>"failedpath_file"
            flush_interval =>0
            path =>"/usr/share/logstash/file.txt"
        }
}
```

在上述配置文件中，定义了使用文件的模式，也设定了重新生成文件的操作（如果它被删除了）。参数 flush_internal 设置为 0，即它接收到每个事件后（event），会把相应的信息输出到文件中。其中也定义了文件路径（path），将在该路径下新建文件（用于存储输出的信息）。

3.9.3　elasticsearch

输出插件 elasticsearch 用于将数据从 Logstash 输出到 Elasticsearch 中。它是最常用的输出过滤插件之一，用于将数据发送到 Elasticsearch 中。如果想在 Kibana 中可视化数据，则需要把数据发送到 Elasticsearch 中。Kibana 使用存储在 Elasticsearch 中的数据来完成可视化。

有关 Elasticsearch 插件的基本配置如下所示：

```
elasticsearch {

}
```

在这个插件中，没有必选的属性项（即：都是可选项）。

这个输出插件有各种各样的可选属性项，但这里只较详细地介绍其中的几种。

- action：对存储在 Elasticsearch 中的文档执行各种操作，如 index（为文档产生索引）、delete（删除指定 ID 号的文档）、create（生成和索引文档，此时 ID 号是唯一标识符）、update（更新指定 ID 的文档）。
- cacert：为.cer 和.pem 文件提供绝对路径，用于在获取信息时对服务器的安全性验证。
- codec：在将数据发送到输出前，编码、解码相应的数据。
- doc_as_upsert：该属性对每一特定文档的更新模式有效。这是一个 upsert 模式。当某个文档有一个已知的 ID 号时，则更新该 ID 号对应的相应文档。如果文档的某个 ID 不存在，则用这个 ID 号新生成一个文档。
- document_id：为文档设定 ID 号，该 ID 作为唯一标识符。一般情况下，该值建议设置为自增的值。

- document_type：提供在文档(document)中将要被存储数据的文档类型(type)。在前面的章节已讲述过，一个索引(index)可以包括多个类型(type)，所以在数据输出时，需要指定是哪些具体类型(type)的索引(index)。
- hosts：为便于和 Elasticsearch 节点进行通信，需要指定主机 IP 地址(端口)。在这里设置的值将要和 Elasticsearch 节点连接并完成发送信息的任务。可以一次指定单个或多个 hosts。默认情况下，这个工作端口是 9200。

> 不要在 hosts 属性中设置 Elasticsearch 的独立主机 master 节点，否则将会向主机 master 节点发送输入信息①。

- index：设置索引(index)的名称，数据将会写入到该索引中。索引名称可以是静态的名称，也可以来自某个字段值，或者是使用了正则表达式的动态的索引名称。
- path：使用代理 proxy 连接 Elasticsearch 节点时需要设置该项的值。在这里应该指定到达 Elasticsearch 节点的路径(path)。
- proxy：指定能连接到 Elasticsearch 节点的代理服务器地址。
- ssl：使得基于 **Secured Socket Layer** (**SSL**)的传输或基于 **Transport Layer Security** (**TLS**)的数据连接交换方式有效。它将会以一种安全通信通道的方式连接到 Elasticsearch 的节点或集群上。

上述设置的值类型以及它们的默认值如下表所示。

设　　置	值　类　型	默　认　值
action	字符串	“index”
cacert	文件路径	无
codec	codec	“plain”
doc_as_upset	布尔	false
document_id	字符串	无
document_type	字符串	无
hosts	数组	[“127.0.0.1”]

① 启动 Elasticsearch 的实例时会启动至少一个节点。节点类型有主(master)节点、数据(data)节点、客户端节点、部落(tribe)节点。对于规模较大、用户较多的集群，master 和 client 在一些极端使用情况下可能会有性能瓶颈，从而使得共存的数据节点发生故障。如果将 master 和 client 独立出来，一旦出现问题，重启后几乎瞬间就能恢复。另外将这些角色独立出来以后，也将对应的计算资源消耗从数据节点中剥离出来，更容易掌握数据节点资源消耗与写入量和查询量之间的联系，便于做容量管理和规划。

续表

设　　置	值　类　型	默　认　值
index	字符串	logstash-%{+YYYY.MM.dd}
path	字符串	/
proxy	<<,>>	无
ssl	布尔	无

配置文件示例：

```
output {
    elasticsearch {
        cacert =>"/usr/share/logstash/cert.pem"
        doc_as_upsert =>true
        document_type =>"elasticsearch"
        hosts =>["localhost:9200","127.0.0.3:9201"]
        index =>"logstash"
    ssl =>true }
}
```

在上面的配置文件示例中，设定了路径，启用了针对文档的 upsert 功能（doc_as_upsert），使用了用于存储输出文件的 document_type。在该配置文件中设定了 Elasticsearch 集群节点 hosts，设定了 Logstash 的 index，用来连接、写入相应的数据。

3.10 学习编解码插件 Codec

编解码插件 Codec 用来编码、解码数据。由于输入数据可以是各种格式的，且在读取数据后可能会将其存储输出到其他不同格式的目的地中，因此需要 Codec 插件。本节介绍其中的一部分 Codec 插件。

3.10.1 rubydebug

rubydebug Codec 是相对简单的一种编解码插件，基于 Ruby Awesome Print 库。它将数据输出到标准 shell 程序的控制台中。

基于 rubydebug 的基本配置文件如下所示：

```
rubydebug {

}
```

本插件没有必选属性。可选属性的设置如下所示：

- metadata：设置向 shell 程序输出信息时，是否包含元数据(metadata)。

其相关的值类型及其默认值如下表所示：

设　置	值　类　型	默　认　值
metadata	布尔	false

配置示例文件如下：

```
codec {
        rubydebug { metadata =>true }
}
```

3.10.2 json

json 编解码插件是一个相对简单的插件，用于编解码基于 json 的数据。

> 如果 json 记录通过/n 分隔，则应该使用 json_lines 编解码插件。

基于 json 的一个简单配置示例如下所示：

```
json {
}
```

在这个插件中没有必选项。可选项如下所示：

- charset：用于设置针对数据的字符编码。该值可以设置为 UTF-8，或者是和输入数据的字符编码方式一样。

相应的属性设置及其默认值如下表所示：

设　置	值　类　型	默　认　值
charset	字符串	UTF-8

配置文件示例如下：

```
codec {
        json { charset =>"UTF-8-MAC" }
}
```

3.10.3 avro

avro 插件读取以 avro 格式输入的数据，并对 avro 格式记录的编码、解码。它常常使用

Kafka 输入,因为 Kafka 是以 avro 压缩数据的格式来发送数据的。Kafka 是一种高扩展性的消息队列,用于处理实时的流数据[①]。

> avro 是一种由社区维护的插件,可能不会出现在 Logstash 的插件中。

有关 avro 的基本配置如下所示:

```
avro {
    schema_uri =>...
}
```

在这个插件中,只有 schema_uri 是必选项(其余均为可选项)。

- schema_uri:用于设定模式(schema)文件,avro 需从中读取数据。该文件可以是本地文件,也可以是以 HTTP URL 表示的文件。

其值类型及其默认值如下表所示:

设　置	值 类 型	默 认 值
schema_uri	字符串	无

配置文件示例如下:

```
codec {
    avro {
        schema_uri =>"/usr/share/logstash/input-schema.avsc"
    }
}
```

3.10.4 multiline

multiline 插件是一种非常重要的 Codec 插件,用于把多个事件(event)合并为一个。使用它,可以把捕获 Java 异常、多个异常事件等合并为一个事件。这在监控 log 日志或执行日志分析时是有用的。借助于各种各样的正则表达式,可以确定哪些行和相应的事件是有关的。

> multiline 过滤已经弃用,推荐使用 multiline 编解码 Codec。

有关 multiline 的基本配置信息示例如下所示:

① 译者注:Kafka 是一种高吞吐量的分布式发布订阅消息系统。

```
multiline {
        pattern =>...
what =>... }
```

这个插件中的 patten 和 what 属性是必选项。

- pattern：设置匹配模式来匹配相应的事件。可以使用正则表达式，或者使用基于 grok 的模式来匹配事件。
- what：定义匹配模式是作为前面的事件还是后面的事件（event）的一部分。它决定了匹配的模式是属于哪部分事件（event）的。其取值可以是 previous 或 next（分别指对应前面还是后面的事件）。

其他可选的配置项如下所示：

- auto_flush_interval：如果能匹配到特定的模式或者没有接收到新的数据，该参数用于设定被转换为事件的时间间隔。如果设定了这个值，则过了设定的时间（秒）后，事件将会被推送至输出端。
- charset：确定数据使用的字符集编码方式。该值被设置为标准的 UTF-8，或设置为和输入数据的编码方式一样。
- max_bytes：在 Logstash 写入俘获的 multiline 事件至输出前，该参数用于设定最大的字节数。
- max_lines：在 Logstash 写入俘获的 multiline 事件至输出前，该参数用于设定最大的文件行数。
- multiline_tag：向 multiline 事件增加 tag，以便匹配相应的模式。
- negate：确定匹配了模式的事件（event）是否是 multiline 事件的一部分，以及在哪个设定下的值将要被使用。
- patterns_dir：设定自定义模式存放的文件夹。可以设定单目录或多目录路径。只有自定义模式时，这个选项才可能用到。

值类型及其对应的默认值如下表所示：

设　置	值　类　型	默　认　值
auto_flush_interval	数值	无
charset	字符串	UTF-8
max_bytes	字节	10MiB
max_lines	数值	500
multiline_tag	字符串	multiline
negate	布尔	false

续表

设　　置	值　类　型	默　认　值
pattern	字符串	无
patterns_dir	数组	[]
what	字符串	无

配置示例如下：

```
codec {
            multiline {
            multiline_tag =>"multiline-event"
            pattern =>"^\["
            negate =>"true"
            what =>"next"
        }
}
```

在上述配置中，设置了 codec 中的 multiline，设置了 tag（其名称为 multiline_event）并在其中标识了所有的 multiline 事件，并特别标明了搜索模式，注明了行是以方括号（[）开始的；注明了 what 取值 next，表明匹配的 multiline 事件应该是下一个事件（next event）中的一部分；negate 取值 true，表明在日志中搜索模式（pattern），直到该模式（pattern）找到。它被认为是单事件中的一部分。

本章曾提到 Logstash 中有超过 200 种由 Elastic 和相关社区开发的插件可供使用。现在，你可能对新插件还有疑惑，即：如果一个新的插件被开发出来并且在最新的 Logstash 中提供，那些使用老版本 Logstash 的用户是否能使用最新的插件呢？

答案是肯定的，可以使用新插件。如果插件与 Logstash 的旧版本插件兼容，就可以安装该插件。下面来看看如何列出插件，如何安装、更新、移除 Logstash 中的可用插件。

3.11　插件的命令行操作

前面已经讨论了各种类型的 Logstash 插件的使用。现在就来学习如何使用 Logstash 安装文件夹 bin 下的 logstash-plugin 命令行，这是用基于脚本命令的方式列表、安装、更新、移除相应插件的方法。

插件的命令行操作方式的命令用法如下：

```
bin/logstash-plugin [OPTIONS] SUBCOMMAND [ARG]
```

上述命令中的 SUBCOMMAND 可以是列表 list、安装 install、移除 remove、更新 update、压缩 pack、解压 unpack 以及生成 generate。

其中，列表 list 用于列出可用的安装插件，安装 install 用于安装某个插件，移除 remove 用于移除某个插件，更新 update 用于更新某个插件，压缩 pack 用于打包某个已经安装的插件，解压 unpack 用于解压某个已经打包的插件。

3.11.1 列出插件列表

使用如下的命令，可以列出在 Logstash 中可用的插件：

```
bin/logstash-plugin list
```

查询有关 list 的各种可用方法，可用如下命令：

```
bin/logstash-plugin list  --help
```

它会显示相关的操作如：--installed， --verbose， --group NAME。

如果给出类似下面这种已经安装的插件的名字（例如下面的 kafka），可列出相应插件的信息：

```
bin/logstash-plugin list kafka
```

采用如下命令，可以显示相应的已经安装插件的版本等信息：

```
bin/logstash-plugin list  --verbose logstash-input-kafaka
```

采用如下命令，能够以分组方式（Input、Filter、Codec）显示已经安装的插件。

```
bin/lostash-plugin list --group filter
```

3.11.2 安装插件

查看有关安装插件 install 中的各种操作，可通过如下命令得到：

```
bin/logstash-plugin install --help
```

上述命令执行后，会显示一些操作项，如：-version，--[no-] verify， --preserve，--development 以及--local。

如果需要安装插件，可使用如下命令：

```
bin/logstash-plugin install logstash-filter-dissect
```

> ℹ 需要拥有根（root）的相关权限来执行针对插件的操作。

如果要安装某个特定版本的插件，可使用如下方式完成：

```
bin/logstash-plugin install --version 1.0.8 logstash-filter-dissect
```

如果安装插件前需要验证相关插件的有效性，可以使用如下方式完成：

```
bin/logstash-plugin install --verity logstash-filter-dissect
```

如果安装插件前无须验证相关插件的有效性，可以使用如下方式完成：

```
bin/logstash-plugin install --no-versity logstash-dilter-dissect
```

如果在本地安装插件，可以使用如下方式完成：

```
bin/logstash-plugin install --local logstash-filter-dissect
```

3.11.3　移除插件

可以使用如下帮助命令，查阅可用的各种移除插件的方法。

```
bin/logstash-plugin remove --help
```

它不包含任何可选用的属性操作。

如果要卸载一个插件，可以使用如下命令方式：

```
bin/logstash-plugin remove logstash-filter-dissect
```

3.11.4　更新插件

可以使用如下帮助命令查阅可用的各种更新插件的方法。

```
bin/logstash-plugin update --help
```

显示可用的操作参数有：--[no-] verrify 和--local。

如果需要更新所有已经安装的插件，可以使用如下命令方式：

```
bin/logstash-plugin update
```

如果需要更新某个特定的安装插件，可以使用如下命令方式：

```
bin/logstash-plugin update logstash-filter-dissect
```

如果需要在更新前验证该插件的有效性，可以使用如下命令方式：

```
bin/logstash-plugin update --verify logstash-filter-dissect
```

如果在更新前不需要验证该插件的有效性，可以使用如下命令方式：

```
bin/logstash-plugin update --no-verify logstash-filter-dissect
```

如果要在本地更新插件，可以使用如下命令：

```
bin/logstash-plugin update  --local logstash-filter-dissect
```

3.11.5 压缩插件

可以使用如下帮助命令来查阅可用的各种压缩插件的方法。

```
bin/logstash-plugin pack --help
```

显示的可用参数有：--tgz，--zip，--[no-] clean 和--overwrite。
如果需要以 GZipped TAR 格式压缩一个插件，可以使用如下的命令方式：

```
bin/logstash-plugin pack --tgz
```

如果需要以 ZIP 格式压缩一个插件，可以使用如下的命令方式：

```
bin/logstash-plugin pack --zip
```

如果需要删除生成的一组插件，可以使用如下的命令方式：

```
bin/logstash-plugin pack --clean
```

反之，如果不删除，则可以使用如下的命令方式：

```
bin/logstash-plugin pack --no-clean
```

如果需要覆盖以前生成的压缩文件，可以使用如下的命令方式：

```
bin/logstash-plugin pack --overwrite
```

3.11.6 解压插件

可以使用如下帮助命令，来查阅各种可用的解压缩插件的方法。

```
bin/logstash-plugin unpack --help
```

显示的可用参数有：--tgz，--zip。
如果需要以 GZipped TAR 格式解压缩一个插件，可以使用如下的命令方式：

```
bin/logstash-plugin unpack --tgz filename
```

如果需要以 ZIP 格式解压缩插件，可以使用如下的命令方式：

```
bin/logstash-plugin unpack --zip filename
```

下面,我们来看看更多的以命令行操作 Logstash 的方式。

3.12　Logstash 的命令行操作

可以用多种不同的命令行操作方式来使用 Logstash。如果要查询更多的基于命令行的操作,可以使用如下帮助命令:

```
bin/logstash --help
```

上述帮助命令运行后,会列出各种可用的操作。我们会在下面继续介绍。

如果需要指定 Logstash 实例的名称,可以使用如下命令方式:

```
bin/logstash --node.name NODENAME
```

如果基于一个配置文件(或者指定配置文件的目录中的配置信息)来使用 Logstash,可以使用如下的命令方式:

```
bin/logstash -f CONFIGPATH
```

或者,使用如下的命令方式:

```
bin/logstash --path.config CONFIGPATH
```

如果直接在命令行中指定一个特定的配置文件来使用 Logstash,可使用如下命令方式:

```
bin/logstash -e "input {stdin {type => stdin} }"
```

如果指定以并行方式运行的进程的数目,可以使用如下命令方式:

```
bin/logstash -w 12
```

或者

```
bin/logstash --pipeline.workers 12
```

上述默认的进程数值是 8。

为了指定运行进程的最大数目来执行过滤 Filter 和输出 Output 的插件连接,可以使用如下的命令方式:

```
bin/logstash -b 50
```

或者

```
bin/logstash --pipeline.batch.size 50
```

上述命令方式的默认数值是 125。

为了在检查新事件前设定等待时间(ms),可使用如下命令方式:

```
bin/logstash -u 10
```

或者

```
bin/logstash --pipeline.batch.delay 10
```

上述应用场景的默认值是 250ms。

如果需要让 Logstash 强行退出关闭(即使它仍在内存中有驻留事件),可以使用如下命令:

```
bin/logstash --pipeline.unsafe_shutdown
```

若需要指定 Logstash 存储数据的目录,可以使用如下命令:

```
bin/logstash --path.data PATH
```

上述命令方式的默认值是:$LS_HOME/data。

如果要设定路径以便找到 Logstash 自定义插件,可以使用如下命令:

```
bin/logstash -p PATH
```

或者

```
bin/logstash --path.plugins PATH
```

> PATH 定义为 PATH/logstash/TYPE/NAME.rb,其中 TYPE 是组名(Input、Output、Filter、Codec),NAME 指插件的名字。

为了在目录中写入 logs 运行 Logstash,可以使用如下命令:

```
bin/logstash -l PATH
```

或者

```
bin/logstash --path.logs PATH
```

为了设定发送到 Logstash 的日志的级别,可以使用如下命令:

```
bin/logstash --log.level LEVEL
```

可用的 LEVEL 取值如下:fatal、warn、error、debug、info(默认值)和 trace。

采用如下命令方式,可以输出编译好的 config ruby 代码(debug log)。

```
bin/logstash --config.debug
```

> 为了能使用--config.debug，必须应用--log.level=debug 这一配置。

使用如下命令，可以显示 Logstash 的当前版本：

```
bin/logstash -V
```

或者

```
bin/logstash --version
```

采用如下的命令行方式，可以验证 Logstash 配置文件是否正确：

```
bin/logstash -f file.conf -r
```

或者

```
bin/logstash -f file.conf --config.reload.automatic
```

> Grok 模式未被验证。只验证句法是否正确。

下面的命令行方式，对 Logstash 配置文件的 auto-reload 有效(enable)：

```
bin/logstash -f file.conf -r
```

或者

```
bin/logstash -f file.conf --config.reload.automatic
```

上述默认值是“false”。

为了检测 Logstash 配置文件是否更新，采用如下的命令行方式，可以设定重新载入的时间间隔(秒)。

```
bin/logstash -f file.conf --config.reload.interval 5
```

上述默认值是 3s。

采用如下命令行，可以指定包含 Logstash 设置文件的目录位置：

```
bin/logstash --path.settings SETTINGS_DIR
```

上述默认值是 $LS_HOME/config。

3.13 使用 Logstash 的小技巧

下面列出和使用 Logstash 有关的一组小技巧。

3.13.1 引用字段及其值

在 Logstash 配置文件中，可以通过名字指定一个字段，随后可以从一个字段向另一个字段中传送一个值。如果想设定 top-level 字段，可以直接使用字段 field 的名称。如果想设定 nested field 字段，可使用[top-level field][nested field]语法形式。对字段而言，Logstash 使用 sprintf 格式对有关字段值进行操作。

sprintf 格式如下：

```
%{ [top-level field] [nested field] …}
```

例如：

```
output {
    elasticsearch {
        document_type =>"%{@version}"
        index =>"logstash_%{type}_%{+YYYY-MM-dd-H}"
    }
}
```

> 索引 index 名不能含有如下特殊字符：\、/、*、?、"、<、>、|、,。另外，索引 index 名必须小写。

3.13.2 添加自定义的 grok 模式

如前所述，我们能够自己生成 grok 模式。这些生成的 grok 模式能直接使用。我们能够使用、修改已经存在了的定义好的 grok 模式，甚至也可以通过正则表达式来自定义新的 grok 模式。

基本的 grok 模式的语法如下所示：

```
%{SYNTAX: SEMANTIC}
```

如果需要使用正则表达式来定义 grok 文件的模式，可以使用如下方式：

```
PATTERNNAME Regular -Expression_Syntax
```

例如：

```
ALPHANUMERIC ([a-zA-Z0-9-]+)
```

如果需要使用一个已经存在的 grok 模式来定义另一个 grok 文件的模式，可以使用如下方式：

```
PATTERNNAME %{EXISTING_GROK-PATTERN}
```

例如：

```
RECORDTIME %{TIME}
```

在本章3.14节“用于解析日志的Logstash配置”中，将会学习更多个性化生成grok模式的方法。

3.13.3　Logstash不显示任何输出信息

这是一个运行Logstash时经常会遇到的问题。在这种情况下，Logstash运行配置文件，但在标准输出设备上无法看到任何输出。这个问题会在一些场合中出现。我们将在如下内容中进行说明。

3.13.3.1　一个输入文件已经被完全读取

在这种情况下，可使用Input插件作为一个文件，指定一个或多个文件读取数据。输入文件使用生成sincedb文件的机制。这样，能够帮助Logstash知道有多少输入文件已经被处理和解析。在这种情形下，如果输入文件已经被完全读取了，那么在Input插件中设置同样的输入文件之后，再次运行Logstash，将不会有任何输出显示。

让我们来看看sincedb文件的内容：

```
948594  0  2049  47484
```

上述各列分别是指：inode(UNIX)或者标识符(Windows)、最大设备编号、最小设备编号、字节偏移，其中：

上述提供的inode列以及最大和最小设备编号，有助于了解基于file输入插件读取的文件。字节偏移决定了在文件中读取的字节数量。

下面是一组针对上述问题的解决方案：

- **方案1**：删除存储在$HOME/.sincedb_*的sincedb文件，再次基于配置文件运行Logstash。此时，将会显示输出信息。
- **方案2**：使用已有设置或者sincedb_path的参数，按如下方法来指定其值：

```
sincedb_path =>"/dev/null"
```

这将会使Logstash认为，它从未解析过输入文件。

> 不推荐使用**方案2**。因为当Logstash重新启动时，完整的文件将会被从头解析，这将导致重复数据的出现。

3.13.3.2 一个输入文件已有一天未更改

在这种情况下，使用 Input 插件作为一个文件，指定一个文件来作为输入源。当输入文件已经有 86400 秒（1 天）没有更改时，Logstash 仍然会运行，但不会再产生任何输出。

针对这种情况，有如下几个解决方案：

- **方案 1**：修改文件，以便它能被 Logstash 读取到。
- **方案 2**：使用 touch 命令（UNIX）修改时间戳（timestamp）。该方案只有在 sincedb 文件的路径是"/dev/null"时，才有效。

3.14 用于解析日志的 Logstash 配置

在本节中，我们将学习如何使用 Logstash 来解析含有不同类型日志的文件，会根据需求使用生成的定制好的 grok 模式来解析数据。

先来看看数据。

日志文件含有数以百万计的涉及 Tomcat 和 Catalina 的日志 log。日志文件也包含应用程序的异常信息 exception、错误信息 error、栈追踪信息 stack trace 等。日志文件含有各种各样级别和日志有关的事件（如 INFO、WARN、ERROR、DEBUG 以及 FATAL 等）。

3.14.1 Catalina 日志示例

首先，看看如下的日志信息：

```
Mar 10, 2016 10:04:37 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 433 ms
Mar 10, 2016 10:04:37 PM org.apache.catalina.core.StandardService
startInternal INFO: Starting service Catalina
```

3.14.2 Tomcat 日志示例

下面是 Tomcat 日志文件示例：

```
    2016-03-10 22:04:40,892  INFO localhost-startStop-1
support.AnnotationConfigWebApplicationContext:208 - Registering annotated
classes: [class matrix.api.config.RESTConfig]
    2016-03-10 22:05:07,248 ERROR http-bio-8080-exec-1
handler.ExceptionHandlerAdvice:57 - We have encountered an internal error.  org.
springframework.dao.EmptyResultDataAccessException: Incorrect result
size: expected 1, actual 0
```

```
at org.springframework.dao.support.DataAccessUtils (Support.java:71)
at org.springframework.jdbc.core.Jdbc.queryForObject(Jdbc.java:489)
at org.impl.HealthCheckDaoImpl.getResult(HealthCheckDaoImpl.java:30))
at org.springframework.web.Handler.invokeForRequest(Handler.java:132)
```

下面生成个性化的 grok 模式，以便匹配 Catalina 和 Tomcat 日志数据。将生成一个名为 grok-pattern 的文件，它位于/usr/share/logstash/patterns 下。

3.14.3 基于 grok 模式的 Catalina 日志

让我们来看看预定义的、定制的、用于解析 Catalina 日志的 grok 模式(pattern)：

```
MONTH
\b(?:Jan(?:uary)?|Feb(?:ruary)?|Mar(?:ch)?|Apr(?:il)?|May|Jun(?:e)?|Jul
(?:y)?|Aug(?:ust)?|Sep(?:tember)?|Oct(?:ober)?|Nov(?:ember)?|Dec(?:
ember)?)\b MONTHDAY (?:(?:0[1-9])|(?:[12][0-9])|(?:3[01])|[1-9])YEAR (?>\d\d)
{1,2} HOUR (?:2[0123]|[01]?[0-9])MINUTE (?:[0-5][0-9])SECOND (?:(?:[0-5][0-9]|
60)(?:[:.,][0-9]+)?)
DURATION (AM|PM)
CATALINA_DATESTAMP %{MONTH} %{MONTHDAY}, %{YEAR}
%{HOUR}:?%{MINUTE}(?::?%{SECOND}) %{DURATION}
JAVACLASS (?:[a-zA-Z0-9-]+.)+[A-Za-z0-9$ ]+
JAVALOGMESSAGE (.*)
LOGLEVEL ([A-a]lert|ALERT|[T|t]race|TRACE|[D|d]ebug|DEBUG|[N|n]otice|NOTICE|[I
|i]nfo|INFO|[W|w]arn?(?:ing)?|WARN?(?:ING)?|[E|e]rr?(?:or)?|ERR?(?:OR)?|[C|
c]rit?(?:ical)?|CRIT?(?:ICAL)?|[F|f]atal|FATAL|[S|s]evere|SEVERE|EMERG(?:
ENCY)?|[Ee]merg(?:ency)?) CATALINALOGLEVEL %{CATALINA_DATESTAMP:datestamp}\s
*%{JAVACLASS:class} %{JAVALOGMESSAGE:loginfo} %{LOGLEVEL:level}: %
{JAVALOGMESSAGE:logmessage}
```

3.14.4 基于 grok 模式的 Tomcat 日志示例

下面是预定义的、定制的、用于解析 Tomcat 日志的 grok 模式：

```
MONTHNUM (?:0?[1-9]|1[0-2])
TOMCAT_DATESTAMP %{YEAR}-%{MONTHNUM}-
%{MONTHDAY}\s*%{HOUR}:?%{MINUTE}(?::?%{SECOND})
REQUEST(?:[a-zA-Z0-9-]+-)+[A-Za-z0-9$ ]+LOGLINE(?:[a-zA-Z0-9-]+.)+(?:[a-
zA-Z0-9-]+\:)+[A-Za-z0-9$ ]+
TOMCATLOG %{TOMCAT_DATESTAMP:datestamp}\s*%{LOGLEVEL:level}
%{REQUEST:request} %{LOGLINE:line} -%{JAVALOGMESSAGE:logmessage}
```

> 通常 Catalina 和 Tomcat 日志的模式是在 grok_pattern 文件中被一次性定义的，因为该文件对于解析这两种模式是通用的。

3.14.5 Logstash 配置文件

下面列出将要用到的 Logstash 配置文件。

```
input {
    file {
        path =>"/usr/share/tomcat/logs/catalina.out"
        type =>"tomcat"
        start_position =>"beginning"
        codec =>multiline  {
            patterns_dir =>"/usr/share/logstash/patterns"
            pattern =>
        "(^%{TOMCAT_DATESTAMP} | ^%{CATALINA_DATESTAMP})"
            negate =>true
            what =>"previous"
        }
    }
}
filter    {
    mutate   {
        gsub =>['message', "n", " "]
        gsub =>['message', "t", " "]
}
grok {
        patterns_dir =>"/usr/share/logstash/patterns"
        match =>[ "message", "%{TOMCATLOG}","message",
"%{CATALINALOGLEVEL}"]
}
date    {
        match =>[ "timestamp", "yyyy-MM-dd HH:mm:ss,SSS","MMM dd,
yyyy HH:mm:ss a"]
        }
    if "_grokparsefailure" in [tags]  {
        drop { }
    }
}
    output    {
```

```
        elasticsearch  {
            hosts =>"localhost:9200"
            index =>"logs"
            document_type =>"logs"
        }
        stdout { codec =>rubydebug }
        if [level] =="ERROR"  {
        email  {
            address =>"smtp.gmail.com"
            port =>"587"
            username =>"gupta.yuvraj"
            password =>"password123"
            use_tls =>"true"
            from =>"<gupta.yuvraj@gmail.com>"
            subject =>"Error status"
            to =>"<kravi.gupta@gmail.com>"
            htmlbody =>"<h2>%{request}</h2><br/><br/><h3>Full
            Event</h3><br/><br/><div
            align='center'>%{message}</div>"
        }
    }
}
```

> 前面提到的 grok 模式以及 Logstash 配置文件均位于：https://github.com/kravigupta/mastering-elastic-stack-code-files/tree/5.1.1/Chapter03。

在前面讲述的配置文件中，已经使用了 Input 插件作为文件，设定了文件名并且给定了用于解析数据的 type，指定了从文件开头来读取文件中的内容。我们使用了 multiline 的 Codec 编码方式，因为 log 日志中含有异常退出(exception)和栈跟踪信息(stack trace)。除了模式 pattern 的目录外，还设定了 pattern 的内容，因为这是一个用户自定义的模式。还应注意到 negate 设为 true，这意味着它将会在日志中搜索模式，直到找到模式，它被认为是单事件的一部分。其中的 what 参数设置为 previous(what => "previous")，表示匹配的 multiline 事件将会是前面事件本身的一部分。

在 filter 插件中，使用了多类型的过滤插件来转换数据。我们已经使用了 mutate Filter 插件，设定所有新行和用 tab 隔开、空格分隔的日志。使用 grok 来指明它们应该匹配的信息字段，通过合适的 patterns_dir(就像在%{TOMCATLOG}和%{CATALINALOGLEVEL }中的那样)自定义 grok 模式。另外，使用 date Filter 告诉 Logstash 匹配已经设定的时间戳(timestamp)字段，在该过滤器的最后，设定不包括未与定义的 grok 模式匹配的任何事件

event。

在 Output 插件中,使用了三种类型的输出,即:输出到 Elasticsearch、输出到标准设备(stdout)、输出到 E-mail。

在基于 elasticsearch 插件的输出中,设定了 Elasticsearch 节点来连接相应的索引 index 和 type 名称。

在基于 stdout 插件的输出中,设定了基于 Ruby Awesome Print Library 的输出日志。

在基于 E-mail 插件的输出中,使用了相应的条件。只有当 level 字段值等于 ERROR 时,该插件才起作用。我们也设定了邮件服务器的相关信息,例如 address 和 port;指定了 username 和 password,以便能基于这些信息连接到邮件服务器。设置了 use_tls 取值为 true,因为要使用该端口来连接并完成加密通信。我们也注意到其中发送 E-mail 的 ID、接受 E-mail 的 ID、邮件主题、发送邮件的内容等。

定位到 Logstash 目录。在命令行中运行 Logstash 配置的方法如下:

```
bin/logstash -f log-config.conf
```

> 如果 E-mail 输出插件丢失,在运行配置文件时,就有可能会遇到错误。可以使用如下方式(需要先定位到 Logstash 文件夹下)来安装 E-mail 输出插件。bin/logstash-plugin install logstash-email-output。

如果通过使用 Logstash 服务来运行 Logstash 配置文件,则需要将 Logstash 的配置文件放到如下目录中。

```
/etc/logstash/conf.d/
```

系统将会运行放置在该文件夹下的配置文件,此时,Logstash 以服务(service)方式运行。

> 运行 Logstash(5.x),将会在 9600 端口打开 Logstash 终端节点,提供可用的 API。

学习了上述内容,可能会对 Logstash 需要打开一个终端节点的 API 的原因感到困惑。我们将在后续内容中进行说明。

3.15 监控系统相应状态信息的 API

本节将介绍目前在 Logstash 中用于检索运行时(runtime)测度信息的各种 API 的用法。

默认情况下,API 绑定在 127.0.0.1,并选个第一个可用端口范围在 9600～9700 进行使用。但是,假设这里有另外一个 Logstash 在 9600 端口运行的实例,就需要指定一个新的运行 Logstash 的端口了。此时,可使用如下的命令方式来完成。

```
bin/logstash --http.port 9602
```

使用如下的命令行方式,可得到有关的基本信息(假设 Logstash 运行在 9600 端口)。

```
curl -X GET http://localhost:9600?pretty
```

针对上述命令的相应返回信息如下所示:

```
{
          "host": "ubuntu",
          "version": "5.1.1",
          "http_address": "127.0.0.1:9600"
          "id" : "6d32ec3e-700b-48eb-bbc3-6a48b2b00f9e",
          "name" : "ubuntu",
          "build_date" : "2016-12-06T13:17:53+00:00",
          "build_sha" : "aa36c4f4b702c6d379e5a97c22627933bca04f68",
          "build_snapshot" : false
}
```

3.15.1　节点信息 API

该 API 能提供相应节点的信息,如节点的操作系统层级信息、节点 JVM 信息、信息管道 pipeline 的相关信息等。

使用如下的命令行方式的 API,可得到所有节点的相关信息。

```
curl -X GET http://localhost:9600/_node/?pretty
```

3.15.1.1　节点的操作系统信息

使用下述的命令行命令,可以得到节点的操作系统级的信息,如操作系统的名称、架构、版本以及可用的进程等。

```
curl -X GET http://localhost:9600/_node/os?pretty
```

可能的返回信息如下:

```
{
"os": {
          "name": "Linux",
```

```
            "arch": "amd64",
            "version": "4.2.0-27-generic",
            "available_processors": 4
        }
}
```

3.15.1.2 JVM 信息

该 API 提供节点 JVM 相关信息(如进程 ID、版本、VM 信息、内容使用、垃圾回收)。

curl -X GET http://localhost:9600/_node/jvm?pretty

上述命令执行后可能的返回信息如下:


```
{
"jvm" : {
            "pid" : 6340,
            "version" : "1.8.0_74",
            "vm_name" : "Java HotSpot(TM) 64-Bit Server VM",
            "vm_version" : "1.8.0_74",
            "vm_vendor" : "Oracle Corporation",
            "start_time_in_millis" : 1481599754165,
            "mem" : {
              "heap_init_in_bytes" : 268435456,
              "heap_max_in_bytes" : 1038876672,
              "non_heap_init_in_bytes" : 2555904,
              "non_heap_max_in_bytes" : 0
              },
            "gc_collectors" : [ "ParNew", "ConcurrentMarkSweep" ]
        }
}
```


3.15.1.3 节点相应的数据管道信息[①]

下述 API 提供节点相应的数据管道(pipeline)的信息,如 workers 的数量、batch_size 大小、batch_delay 的设置值以及和配置有关的信息(如 config_reload_automatic 和 config_reload_interval 的取值等)。

curl -X GET http://localhost:9600/_node/pipeline?pretty

① 译者注:原文 pipleine 有误,此处应为 pipeline。

返回结果如下：

```
{
    "pipeline" : {
            "workers" : 4,
            "batch_size" : 125,
            "batch_delay" : 5,
            " config_reload_automatic" : false,
            " config_reload_interval" : 3
    }
}
```

3.15.2　插件信息 API

通过使用下面介绍的 API，可以提供所有安装在 Logstash 中的插件的信息。使用方法如下所示：

```
bin/logstash-plugin list --verbose
```

运行如下命令，可以得到所有插件的信息。

```
curl -X GET http://localhost:9600/_node/plugins?pretty
```

执行上述命令后可能的返回信息如下所示：

```
{
    "total": 94,
    "plugins": [
    {
        "name": "logstash-codec-cef",
        "version": "4.1.0"
    },
    {
        "name": "logstash-codec-collectd",
        "version": "3.0.2"
    },
    {
        "name": "logstash-codec-dots",
        "version": "3.0.2"
    },
    ⋮
]
```

3.15.3 节点状态 API

使用本节介绍的 API，可以提供节点运行时（runtime）统计信息，如节点的 JVM 状态、处理状态、内容使用状态以及数据管道（pipeline）的状态等。

使用如下的命令，可以马上得到所有节点的相关信息。

```
curl -X GET http://localhost:9600/_node? status? pretty
```

3.15.3.1 JVM 状态

通过使用下述命令，可以得到有关节点的线程、存储空间使用、垃圾回收等信息。通过使用如下命令，能得到 JVM 状态。

```
curl -X GET http://localhost:9600/_node/status/jvm?pretty
```

3.15.3.2 进程状态

使用如下 API，可以得到有关文件的描述信息、存储空间信息、CPU 信息、进程状态信息等。

```
curl -X GET http://localhost:9600/_node/status/process?pretty
```

3.15.3.3 数据管道 pipeline 状态

使用如下 API，可以得到数据管道事件（pipeline event）、插件、重载信息以及数据管道（pipeline）状态等。

```
curl -X GET http://localhost:9600/_node/status/pipeline?pretty
```

3.15.4 Hot threads API

使用本节介绍的 API，可以列出所有和运行 Logstash 有关的 hot thread 信息。这里的 hot thread 是指那些使用了较高 CPU 资源且长久驻留的 Java 线程。

运行如下命令，可以得到相关的 hot thread 的信息。

```
curl  -X GET http://localhost:9600/_node/hot_threads?pretty
```

下面列出和 hot thread 相关的参数。

3.15.4.1 线程

运行下述的命令，可以得到线程 thread 的数量。默认情况下，线程数量设置为 3，也可

以使用如下命令来设置线程信息。

```
curl -X GET http://localhost:9600/_node/hot_threads?threads=1
```

3.15.4.2 Human

如果 human 的值设置为 true，则会以纯文本而非 JSON 格式提供相应信息。默认状态下，该值设置为 false。可以使用如下命令完成相应的设置。

```
curl -X GET http://localhost:9600/_node/hot_threads?human=true
```

3.15.4.3 忽略空闲线程：ignore_idle_threads 参数

如果 ignore_idle_threads 参数设置为 true，则不会提供空闲进程的响应。默认状态下，其值为 true。可以使用如下命令来忽略空闲进程。

```
curl -X GET http://localhost:9600/_node/hot_threads?ignore_idle_threads=false
```

3.16 本章小结

本章介绍了 Logstash 的用法、应用需求及其特点，内容涉及基本的 Logstash 数据流及 Logstash 配置架构；详细介绍了各种插件及其用法，涉及插件的命令行操作及其用法；还介绍了一些小技巧，以及 Tomcat 和 Catalina 日志的应用实例；并且介绍了如何生成、运行一个 Logstash 配置文件来执行面对日志的 ETL 操作。在本章最后，还介绍了面向终端用户的各种监控 API 的用法，以便能得到 Logstash 的相关信息（有时，Logstash 看起来像一个充满了秘密的黑盒子）。

下一章将会讲述 Kibana 及其特点，并介绍 Kibana 的各种操作界面。



Kibana 界面

学习 Elasticsearch 和 Logstash 之后，从这一章开始将学习 Kibana，它提供了为收集和存储的数据执行可视化的界面。在这一章中，将主要学习 Kibana 的界面和所有重要的细节，还将学习基于 Lucene 查询的搜索、聚合等，并了解它们所扮演的角色。

在本章的末尾，将了解 Kibana 的新特性、各种选项卡及其组件的功能，了解如何创建可视化、面板和自定义 Kibana 设置，也将学习如何将数据插入到 Elasticsearch，创建索引模式并可视化数据，以及如何创建、共享和嵌入这些可视化的内容。

本章主要内容如下：

- Kibana 及其功能；
- 探索 Discover 界面；
- 查询和搜索数据；
- 探索 Visualize 界面；
- 了解聚合；
- 探索 Dashboard 界面；
- 了解 Timelion 时间线；
- 探索开发工具 Dev Tools；
- 探索 Management 界面；
- 综合应用。

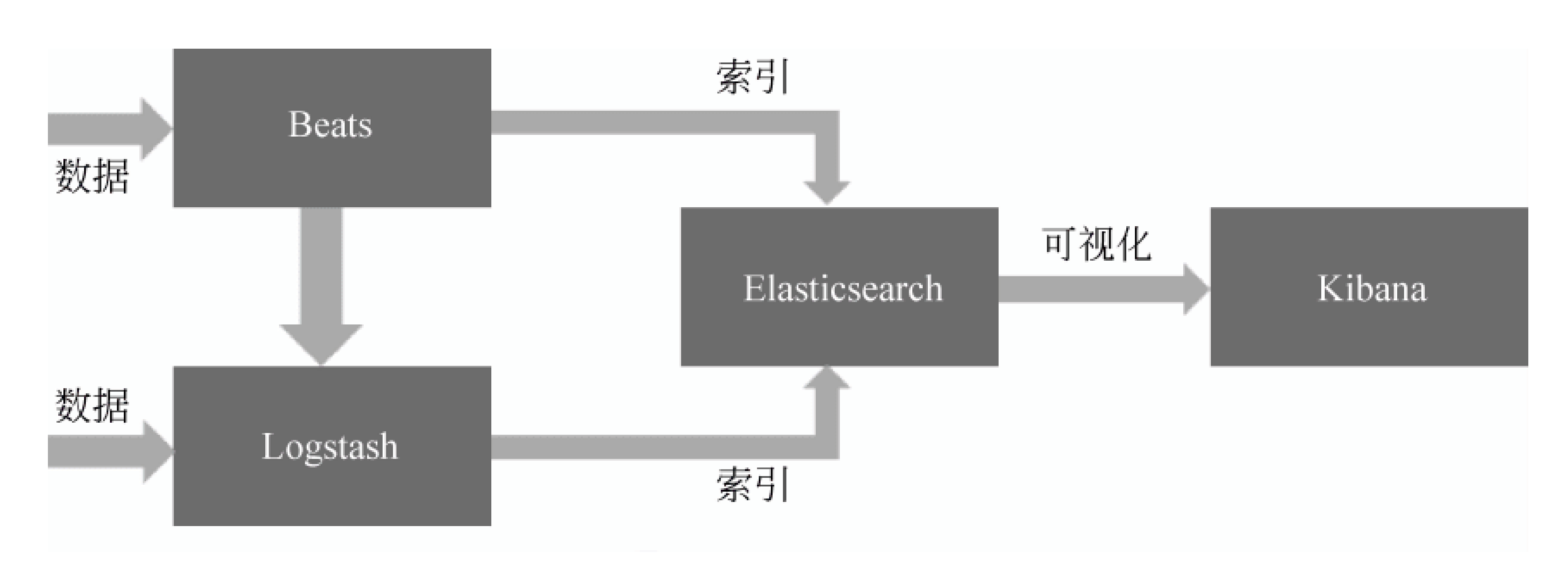
4.1 Kibana 及其功能

Kibana 是可视化工具，它用于可视化存储在 Elasticsearch 中的数据（结构化或非结构化数据）。作为 Elasticsearch 的顶层工具，Kibana 用于搜索、查看、分析数据，它还可以对存储在 Elasticsearch 索引中的数据执行可视化。Kibana 和 Elasticsearch 都是 Elastic 的产品，二者都使用了 Apache 许可证，它们基于 Apache Lucene，使用了 Elasticsearch 强大的功能（Lucene 查询语法、聚合）。Kibana 提供了数据分析的功能，将数据可视化为不同的图表

形式，如面积图、折线图、条形图、饼图、瓦片地图、数值统计和数据表，并创建可共享或嵌入的面板。面板提供了一种统一的方法来在同一位置陈列所有可视化的内容。所有的搜索、可视化、面板都以 JSON 格式存储为底层 JSON 文件。所有搜索、可视化、面板的 JSON 文件都存储在 Elasticsearch 中。Elasticsearch 可以实时处理和分析大量的数据，在自动更新功能的帮助下，可视化/面板将产生动态变化，这种变化提供了实时的画面展示。通过基于浏览器的界面，可以很简单地搭建和使用 Kibana。

Kibana 需要一个 Web 服务器来工作(已整合在 Kibana4 及后续版本中)，它可以在任意支持工业标准的现代浏览器中无缝地运行，对工业标准的支持可以使不同的 Web 浏览器渲染出同样的内容。Elasticsearch 中开放了基于 REST 的 API 接口，Kibana 则使用这种接口与 Elasticsearch 进行交互，Elasticsearch 提供简单的 HTTP 响应来请求和接收信息。REST 是一种无状态协议，它使用 HTTP 协议来完成多种系统之间的交流，而非使用复杂的 CORBA、SOAP、WSDL、RPC 等方式实现。REST 使用 HTTP 协议，并提供 CRUD(创建、读取、更新和删除)操作。

下面是第 1 章 Elastic Stack 概述中的图，这张图展示了 Elastic Stack 的典型高层架构。

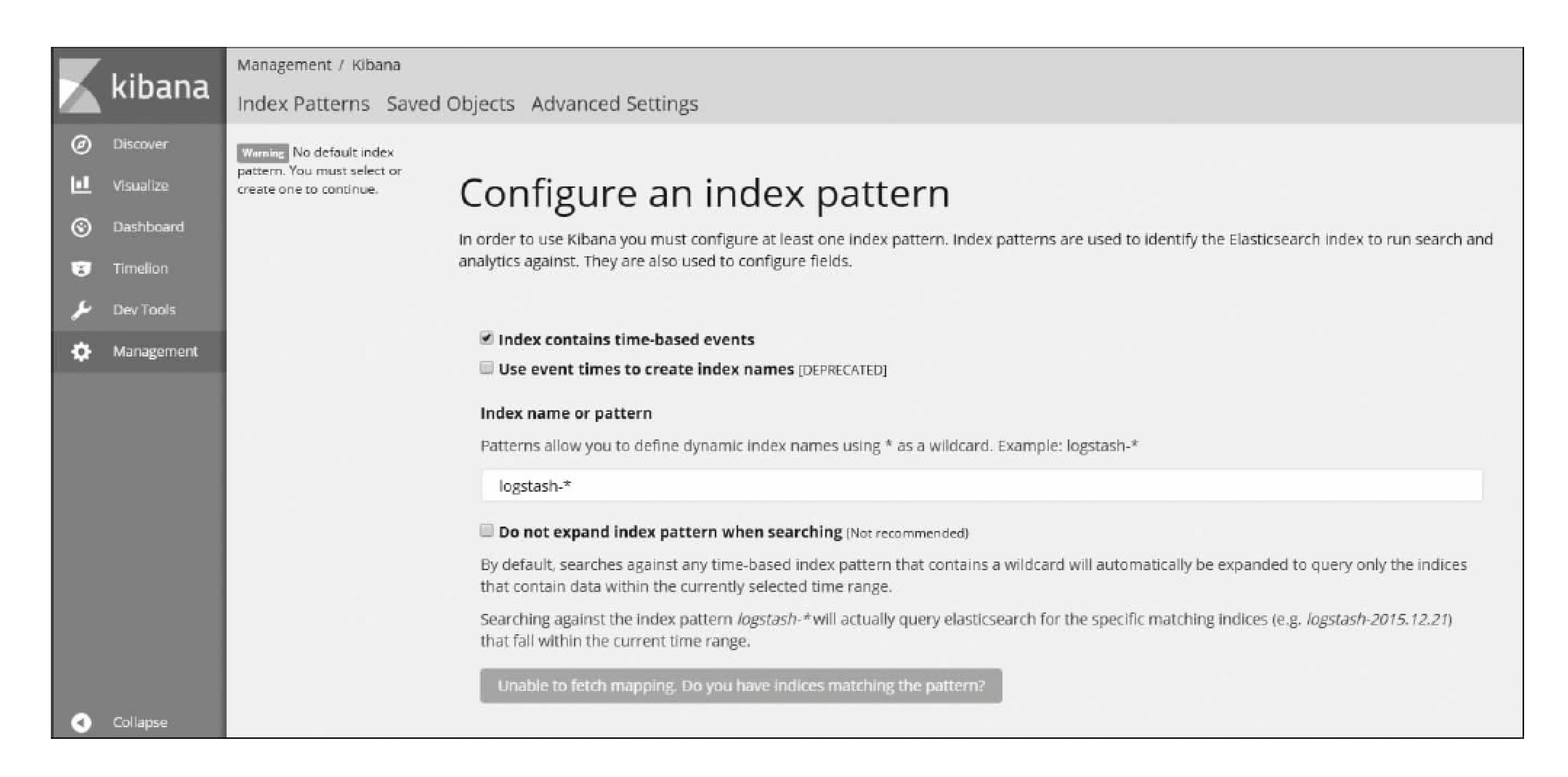


从该图中可以观察到，Kibana 对用户来说是一种终端。无论通过 Logstash 或者 Beats 收集的数据，还是储存在 Elasticsearch 中的数据，都可以通过 Kibana 的界面来查看和分析。

Kibana 的界面如下图所示。

这个界面相较于早期版本的 Kibana 来说，有一个受欢迎的改变，它带来了一个全新的用户界面。这个界面已经按照要求重新设计，并保持了简约的风格。可以通过展开/折叠左边的窗格来展示名称和图标，或者仅展示各个选项卡的图标。这个界面主要由左侧窗格中的 6 个选项卡组成，描述如下：

- **Discover(查询)**：这个页面提供了数据的概述，比如所选索引的名称、数据中的字段列表和存储在字段中的文本数据。它提供了对数据的搜索、查询、过滤、分析和显示搜索结果等功能。
- **Visualize(可视化)**：这个页面提供多种类型的可视化概览，也提供从已保存的或新的搜索中创建可视化的若干步骤，还可以打开已保存的可视化文件。它也提供创



建、查看、更新和删除自定义可视化内容的功能。

- **Dashboard（面板）**：这里为所有已保存的可视化和搜索提供了一个单独的视图页面。它提供了对可视化内容的编辑、移动、调整大小和删除等功能。它通过组合多个可视化并且同时显示，来使你更清楚地了解数据。
- **Timelion（时间线）**：这个页面是最新添加到 Kibana 界面中的，它用于可视化时间序列。受到“时间轴”这个词的启发，它可以将多种数据来源关联到一个基于时间序列实现可视化。它使用了简单易用的表达式来访问数据，这将构成最初的学习曲线。它诞生于 Elastic 实验室的研究项目，这些项目专注于研发未来的产品。
- **Dev Tools（开发工具）**：作为 Kibana 的一部分，开发工具由一个控制台组成。控制台提供了简单且直观的用户界面，通过使用 Web 浏览器与 Elasticsearch 的 REST API 实现交互。开发工具由分析器 Profiler 组成，配置文件可以作为 X-Pack 插件的一部分来使用。该组件将在第 9 章中的了解 Profiler 一节中进行介绍。
- **Management（管理）**：这个页面提供了 Kibana 中使用的所有的配置的概述。它用于配置单个或多个新的索引模式，修改现有的配置参数，导出和导入已保存的对象（搜索结果、可视化和面板），显示 Kibana 版本的详细信息，创建/提交 Kibana 中的内容。

4.2 探索 Discover 界面

通过对索引文档的分析，Discover（查询）页面可以帮助你轻松地处理数据。它支持对数据执行不同类型的搜索，有助于理解数据的含义，以及使用数据来创建可视化的统计图表。查询界面还提供了在不离开查询页面的条件下，通过切换索引模式来选择不同索引名

称的功能，用户可以简单地执行搜索查询，使用过滤器并且查看查询和过滤匹配的文档。

Discover 界面如下图所示：



查询页面中的组件如下：

- **Time Filter（时间过滤器）**：过滤特定时间范围内的数据。
- **Search Box（搜索框）**：用于搜索和查询数据。
- **Toolbar（工具栏）**：包含新建搜索、保存搜索、打开已保存的搜索和共享的选项。
- **Index Name（索引名称）**：显示所选索引的名称。
- **Fields List（字段列表）**：显示所选索引中的所有字段的名称。
- **Number Of Hits（命中数量）**：按指定的时间间隔显示文档总数，并与搜索查询匹配文档结果相对应。
- **Histogram（柱状图）**：显示每次查询结果中的文档数量的信息。
- **Documents Data（文档数据）**：显示包含完整数据的文档。

> ℹ 只有索引中包含基于时间的事件时，柱状图才可以在 **Discover** 界面中显示。

下面分别介绍每个组件。

4.3 时间过滤器

时间过滤器（Time Filter）可以根据指定的时间范围来钻取分析数据。它可以灵活地在任何时间，根据需要搜索、查询、过滤数据，并且能非常简单地分析特定时间的数据，或者根据要求自定义时间周期。同时，在时间过滤器面板中还提供了各种设置选项。

> 默认情况下，时间过滤设置为最近 15 分钟，这意味着它只能显示前 15 分钟的数据。

单击时间过滤器后，可以找到如下几节中提到的各种子选项。

4.3.1 快捷时间过滤器

快捷时间过滤器(Quick Time Tilter)提供众多的时间范围选项，来方便快捷地查看今天、本周、今年、上周、前一个月、最近 30 分钟、最近 24 小时、最近 90 天、最近五年等时间范围内的数据。

4.3.2 相对时间过滤器

相对时间过滤器(Relative Time Filter)用于根据相对时间对数据进行过滤。它提供了 From 和 To 的输入框，依靠它们来指定从现在开始的相对时间。指定相对时间的各种选项包括秒、分钟、小时、日、星期、月和年。

它还提供了一个小的复选框，用来将相对时间舍入到与其最接近的秒、分、小时、日、星期、月和年。

4.3.3 绝对时间过滤器

绝对时间过滤器(Absolute Time Filter)可以根据时间和日期的格式，提供来自 From 字段的起始日期/时间，以及来自 To 字段时间范围的结束日期/时间。时间和日期的格式是 YYYY-MM-DD HH：mm：ss.SSS，解释如下：

- YYYY 定义年份为四位数(比如 2017)；
- MM 定义月份为两位数(从 01 到 12)；
- DD 定义天为两位数(从 01 到 31)；
- HH 定义小时为两位数(从 00 到 23)；
- mm 定义分钟为两位数(从 00 到 59)；
- ss 定义秒为两位数(从 00 到 59)；
- SSS 定义毫秒为三位数(从 000 到 999)。

4.3.4 自动刷新

自动刷新(Auto-refresh)用于在一定时间间隔后，自动刷新页面，时间间隔可以为 5 秒、1 分钟、12 个小时、1 天等。它主要用于分析实时流数据，因为它提供了最新的数据。单击时间过滤器后，**自动刷新**选项会出现在时间过滤器的左边。

时间过滤器的设置也可以通过单击柱状图中的任意竖条来进行，或者也可以通过单击和拖动鼠标指针来选择时间范围。

4.4 查询和搜索数据

搜索框用于执行匹配文档的多种类型的查询。搜索时，整个查询界面及相关组件会自动执行刷新。Kibana 底层使用了强大的 Lucene 查询语法来查询数据。因为 Kibana 使用了 Elasticsearch 的底层功能，所以其 Lucene 查询提供了执行多种搜索的能力，这些搜索包括了从简单到复杂查询的各种类型。

Lucene 查询中提供了许多搜索数据的方法，让我们逐个来看一下。

4.4.1 全文检索

全文检索(full-text search)可以实现在一篇完整的文本中对某个词项进行搜索。各种全文检索方式如下：

- 搜索单个词项：

例如：搜索一个单独的词项，在搜索栏中输入 kibana。

- 搜索一个短语(词组)：

例如：搜索一个短语，在搜索栏中输入 elasticsearch kibana。

> 默认情况下，对短语的搜索使用布尔表达式 OR 来执行。

- 搜索一个确切的词项或者短语：

例如：搜索一个确切的词项或者短语，需要在搜索词两端加上双引号，在搜索栏中输入 elasticsearch kibana。

- 在特定的字段中搜索词项：

例如：搜索一个字段中存在的词项，需要先输入字段名，接着输入一个英文冒号，后面跟一个词项，例如 text：kibana。[①]

4.4.2 范围搜索

执行范围搜索(range search)，通用的句法是“字段名称：{ 起始范围 TO 结束范围 }”或者是“字段名称：[起始范围 TO 结束范围]”。TO 是一个必须使用的关键字，并且必须

① 译者注：此处的 4 个例子，在原文中第 1、2 个和第 3、4 个的位置分别是互换的，笔者认为这里存在错误，在本书中将其以正确的位置放置。

用大写字母来指示搜索范围：

- 日期范围搜索：例如，搜索一个日期范围，输入 modified_date：[2016-04-01 TO 2016-06-30]。
- 数字范围搜索：例如，搜索一个数字范围，输入 no_of_tweets：[100 TO 1000]。
- 字符串范围搜索：例如，搜索文本范围，输入 text：[elasticsearch TO kibana]。

> 方括号[]提供了包含边界值(inclusive)的范围查询。也就是说，搜索结果包含指定的搜索关键字。花括号{}提供了不包含边界值(exclusive)的范围查询，也就是说，查询结果不包含指定的搜索关键字。

4.4.3 布尔搜索

布尔搜索(boolean search)是提供操作符的搜索类型，如下所示：

- OR 操作符：用于组合多个项并搜索其中任意项。
  例如：输入 elasticsearch OR kibana，其中 OR 是关键字。
  还可以通过输入 elasticsearch || kibana 来使用 OR 操作符，在这里||是关键字。
  同样，在指定多个单词时，OR 是默认的操作符。比如 elasticsearch kibana 表示 elasticsearch 或 kibana。
- AND 操作符：用于组合多个项并搜索所有项。
  例如：输入 elasticsearch AND kibana，其中 AND 是关键字。
  还可以通过输入 elasticsearch &&kibana 使用 AND 操作符，在这里 && 是关键字。
- NOT 操作符：用于在搜索结果中排除 NOT 后面的项。
  例如：输入 elasticsearch NOT kibana，其中 NOT 是关键字。

4.4.4 邻近搜索

邻近搜索(proximity search)用于寻找彼此之间有一定距离的项，它使用波浪线(～)作为关键字。

比如，输入："java elasticsearch"~50，搜索结果将提供 Java 和 Elasticsearch 之间最大距离为 50 字的文档集。

4.4.5 通配符搜索

通配符搜索(wildcard search)用于搜索使用通配符表达式的匹配模式，如下所示：

- 单个字符搜索：用于在通配符搜索中替换单个字符。

例如：输入？ac，会得到 mac、sac 和 tac 这样的搜索结果。其中的问号(?)是关键字。

- 多个字符的搜索：用于在通配符搜索中替换 0 个或者多个字符。

  例如：输入 m * c，会得到 mac、magic、macintosh、machine、music 这样的搜索结果。其中的星号(*)是关键字。

4.4.6 正则表达式搜索

正则表达式搜索(regular expressions search)用于查找特定模式下的词项、短语或字段。它使用斜杠(/)和方括号([])[①]来指定匹配模式。

例如，输入/mu[dgm]/ AND /ma[cdnp]/，搜索结果中的文档中包含两种词项的结合，第一个种是 mud、mug 和 mum 这样的词项，第二种是 mac、mad、man 和 map 这样的词项。

4.4.7 分组

分组(grouping)通过结合多个词项、短语和不同类型的操作符来创建复杂查询。

例如：输入(elasticsearch OR kibana) AND(Logstash OR "Lucene query")。

4.5 字段和过滤器

字段列表用于提供数据中使用的所有字段的列信息。它有助于分析数据集，并且可以据此了解文档中都有哪些字段，以及字段中包含什么类型的信息。字段分为选定字段和可用字段，其中每种字段名称按照字母顺序排列。

它还提供三个选项，包括了解现有字段中的数据、字段中的前五个值以及对包含该值的文档的百分比划分。这些信息可以通过单击字段列表中的任意字段来获取。同样，也可以通过单击 **Add** 添加字段，查看所有文档该字段的数据，该字段位于字段名称旁边。此时程序会将指定的字段名移动到已选定的字段列表中，从而提供字段当前值的信息。

4.5.1 过滤字段

对字段的过滤操作可以基于字段的值筛选搜索结果来完成，字段搜索可以使用的字段的过滤间接完成。两种过滤器形式如下：

- 正过滤器(positive filter)：用加号(＋)表示，它可以展示包含字段值的文档。

① 译者注：在正则表达式中，方括号[]内部的内容虽然是连续的字符串，但其中各个字符之间是“或”的关系。

- 负过滤器(negative filter):用负号(—)表示,它可以排除所有包含字段值的文档。

过滤器可以通过以下两种方式添加:

- 单击字段名称(位于字段列表下)添加正或负过滤器,并且选择想包含或者排除的字段值。
- 展开数据并选择要包含或排除文档的值,接着单击文档数据中的字段值来添加正或负过滤器。

添加过滤器后,下面介绍过滤器的功能。

4.5.2 过滤器的功能

过滤器提供了一个简单的方式,来根据需要过滤并分析数据。它可以通过应用不同的过滤方式来获取数据,如下所示:

- **启用过滤器(Enable filter)**:启动过滤器并且显示匹配过滤器的文档。它类似于正过滤器并且始终以绿色颜色显示。
- **禁用过滤器(Disable filter)**:禁用过滤器,直接显示所有文档而不进行过滤。它以条纹的样式显示。
- **锁定过滤器(Pin filter)**:将过滤器锁定在 Kibana 页面上。例如,在 Discover 页面使用并锁定过滤器,再进入 Kibana 的任何其他任何页面,过滤器依然会出现。
- **解锁过滤器(Unpin filter)**:解锁当前正在固定的过滤器。解锁后,这些过滤器不会在 Kibana 页面上继续存在。
- **反转过滤器(Invert filter)**:用于匹配的过滤器被转换后,会显示原本不被匹配的文档。再次反转后,又将显示匹配的文档。
- **过滤器状态切换(Toggle filter)**:切换过滤器的状态。启用中的过滤器以绿色表示,而禁用的过滤器则以红色表示。在切换过滤器的状态时,启用中的过滤器会被切换到禁用状态,反之亦然。
- **删除过滤器(Remove filter)**:将过滤器彻底删除。该操作执行后,当前已添加的过滤器将被删除。
- **自定义过滤器(Custom filter)**:对添加的过滤器进行自定义设置,其中提供了过滤器的 JSON 表示,可供用户自定义。它还可以为重复利用的过滤器提供别名。

之前提到的所有过滤器均可以在 Kibana UI 中查看,如下图所示:



4.6 查询页面选项

在工具栏和搜索框中还有许多其他选项，例如：

- **新建搜索(New Search)**：清除已有的搜索查询，并创建一个新的搜索查询。
- **保存搜索(Save Search)**：保存搜索结果和选定的索引。
- **打开已保存的搜索(Open Saved Search)**：加载已保存的搜索查询和指定索引。它用于打开已保存的搜索查询和索引。
- **分享(Share)**：通过链接来分享搜索查询，这里也可以生成短链接来直接分享搜索结果。它也可以用于分享已保存的搜索结果。
- **向文档数据中添加字段(Adding fields to the Document Data)**：将鼠标指针悬停在字段列表上方时，对应位置的字段名后面会出现一个 **Add** 按钮，单击该按钮可以轻松地把字段添加到文档数据中。
- **在文档数据中删除字段(Removing fields from the Document Data)**：将鼠标指针悬停在要删除的字段列表上方时，对应位置的字段名后面会出现一个 **Remove** 按钮，单击该按钮可以轻松地把字段从文档数据中删除。
- **查看数据(Viewing Data)**：单击位于文档数据起始列左侧的 ▸ 按钮，可以轻松查看文档中的数据。
- **文档排序(Sorting Documents)**：单击列名称旁边的排序按钮，可以对文档数据中已添加的字段进行排序。
- **移动字段(Moving fields)**：单击删除字段标志旁边的(>>)或(<<)，可以对文档数据中已添加的字段重新排序，并可将其移动到文档数据的左侧或右侧。

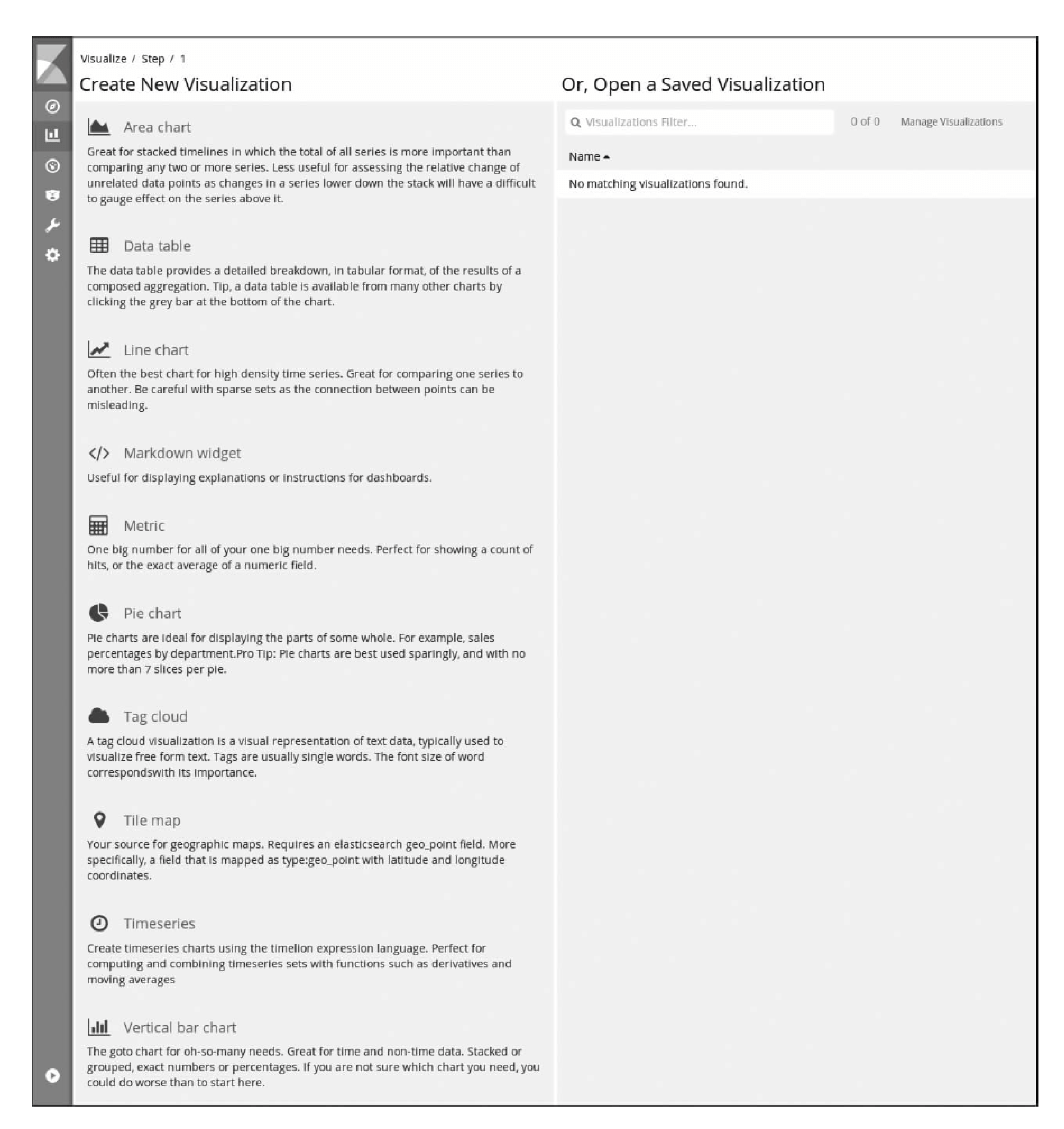
4.7 探索 Visualize 界面

可视化是 Kibana 的核心，也是创建 Kibana 的重要原因之一，它的出现为用户提供了可视化大量数据的能力。随着 Elasticsearch 和 Logstash 的日益普及，如果对 Elasticsearch 中存储的数据缺少直接进行可视化的能力，就会导致用户不能理解数据中所包含的信息。Kibana 的出现，解决了这个问题，并且提供了简单且直观的界面。它克服了近乎实时的情况下可视化大量数据的挑战，其核心组件使 Kibana 成为一款功能丰富的开源软件。

可视化页面有助于对 Elasticsearch 中存储的所有数据实现可视化。使用 **Discover** 界面了解数据之后，可视化页面能轻松创建可视化统计图表。可视化有助于理解数据，而不是浏览大量可能没有任何意义的数据。**可视化**页面提供了创建、查看、修改和删除自定义可视化

内容的功能。

可视化页面通常如下图所示：



首次使用可视化界面时会有些棘手，因为可视化的设计需要如下四个步骤：

（1）选择可视化类型，创建可视化统计图表；

（2）选择索引的名称；

（3）选择搜索数据源；

（4）可视化画布。

步骤（1）、（2）、（3）都非常简单，这三个步骤分别提供了对应的选项，可以选择可视化类型，选择索引，以及决定是否使用已保存的或新的搜索数据源。步骤 4 打开了一个新的界面，用户对于这个界面最初可能较为陌生。在创建、编辑、配置和删除可视化内容方面，可视化画布都是核心部分。可视化画布的界面如下图所示：



默认情况下,时间过滤器设置为最近 15 分钟。

可视化画布包括三个主要部分:

- 工具栏(标记为数字 1);
- 聚合设计(标记为数字 2);
- 预览可视化(标记为数字 3)。

在了解创建可视化的方法前,先来了解 Elasticsearch 聚合的底层组件,这些组件被用于创建可视化内容。

4.7.1 了解聚合

可视化以统计图表和地图等表现形式,利用 Elasticsearch 聚合提供简单或复杂的数据表示。聚合形成了在 Kibana 中创建可视化的引擎,并且它遵循 Elasticsearch 中的聚合逻辑。聚合由 Elasticsearch 的 facet 模块演化而来,它只负责收集数据的任务,支持数据的快速查询和简单聚合。聚合主要分成两类:bucket 聚合和 metric 聚合。

4.7.1.1 bucket 聚合

bucket(桶)聚合可以根据一个或多个标准在不同的 bucket 中分配不同的文档。它用于各种文档分组,并且以一个标准进行评估,来决定哪个文件匹配哪个 bucket。每当要进行聚合时,所有的文档都需要通过评估找到能承载自己的 bucket,所以直到所有的文档都能被评估,上述过程才能执行。它也可以计算 bucket 中的文档,并返回每个 bucket 中存在的文档数量。

bucket 聚合提供了灵活性,它可以与 metric 聚合相结合,也可以创建子聚合。子聚合

可以计算根据父聚合生成的每个 bucket 中的文档。同样，不同 bucket 聚合策略是不同的，可使用单个 bucket 策略，可使用多 bucket 策略，还可在聚合数据时动态创建 bucket。bucket 代表了可视化的横坐标。

在 Kibana 中不同的 bucket 聚合方式如下：

- **Date Histogram（日期柱状图）**：此聚合用于数据类型为日期的字段，如果索引包含基于时间的事件，则由 Kibana 自动提取。它需要一个具有日期类型的字段和一个区间参数来聚合数据。它将符合 bucket 聚合标准的文档分组，而 bucket 聚合的标准由区间参数指定。可用的区间参数有自动、秒、分钟、小时、日、周、月、年和自定义。

  比如，文档包含多个日期的字段。如果指定日期类型字段和区间参数，比如按月选取，之后，所有的文档都会按照月进行分组，并且包含从 7 月 1 日到 7 月 31 日数据的文档将送入 7 月的 bucket 中，以此类推。

- **Histogram（柱状图）**：此聚合用于包含数字或数据值的字段。它需要一个数据类型编号的字段和任何有数据的值，来提供聚合数据的区间。它能基于指定的时间间隔动态创建 bucket 聚合。它把符合 bucket 聚合标准的文档分组，而这样标准由区间参数指定。

> Histogram 聚合与 Date Histogram 聚合类似。不同之处在于，Histogram 使用数字类型的数据，而 Date Histogram 使用日期类型的数据。

  例如，对于包含 1～1000 数值字段的文档，它的间隔被指定为 100。它将聚合所有的文档并且基于 bucket 的值分组，bucket 的值的关键字为 100、200，以此类推。

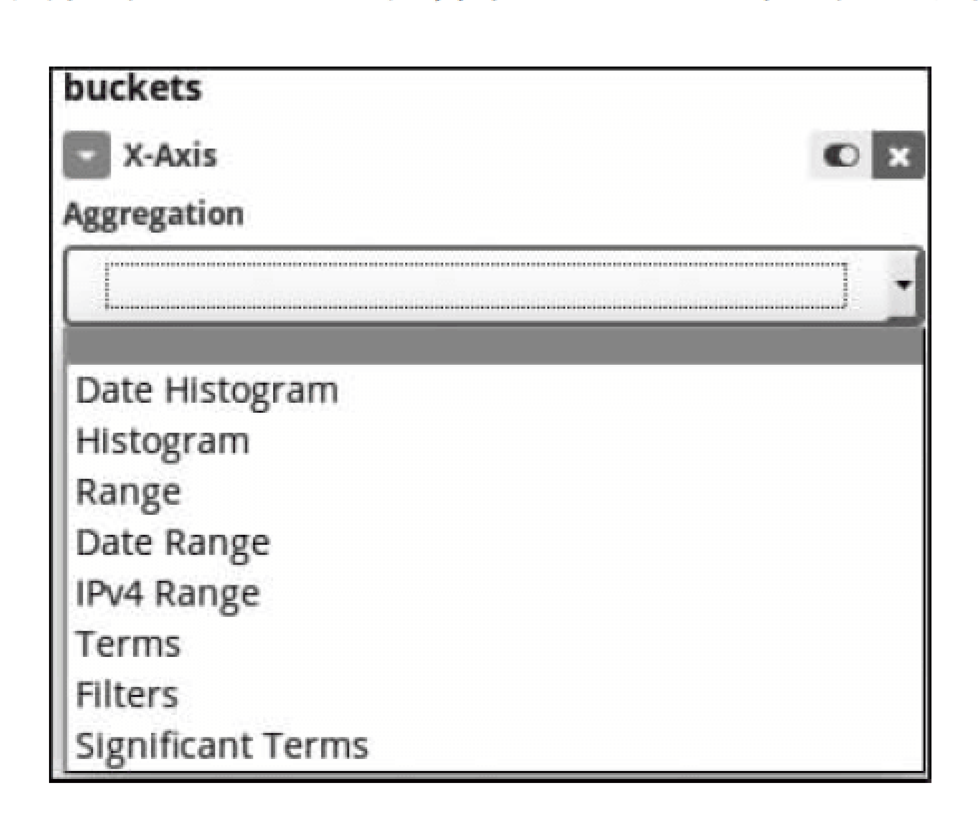
- **Range（范围）**：此聚合用于包含数值的字段。它需要一个数据类型的字段，区间需要以数值的形式指定来聚合数据。它基于指定的范围创建 bucket 聚合，并把符合 bucket 聚合标准的文档分组，而这样的标准由范围指定。

> Range 聚合与 Histogram 聚合类似。不同之处在于，后者可以动态创建 bucket 聚合，而前者以 From 和 To 之间的日期范围指定间隔。

- **Date Range（日期范围）**：此聚合用于包含日期类型值的字段。它需要一个日期数据类型的字段，并且用来聚合数据的日期间隔需要以程序接受的日期格式来指定。它基于指定的范围创建 bucket 聚合，并把符合 bucket 聚合标准的文档分组，而这样的标准由范围 range 指定。

> Date Range 聚合与 Date Histogram 聚合类似。不同之处在于 Date Histogram 聚合动态创建 bucket 聚合，而 Date Histogram 聚合的 bucket 以 From 和 To 之间的日期范围来指定，而且日期范围包括 From 的值，但不包括 To 的值。

- **IPv4 Range（IPv4 范围）**：此聚合用于包含 IP 类型值的字段。它需要一个 IP 数据类型的字段，其用来聚合数据的范围也需要进行指定。IPv4 Range 基于所提供的 IP 范围来创建 bucket 聚合，它把符合 bucket 聚合标准的文档分组，而这样的标准由范围指定。
- **Terms（词项）**：此聚合基于文档中字段的值动态创建 bucket 聚合。它与 SQL 语句中的 GROUP BY 类似。它以字段值创建 bucket，并且将包含该字段的文档放入其中。它允许依据 Top N 或者 Bottom N 排列字段。例如，文档中包含一个以国家命名的字段。可依据国家名称将数据分组来找到前五个或者后五个国家。
- **Filters（过滤）**：此聚合基于搜索查询创建 bucket，它使用搜索查询作为过滤器，同时也是 bucket 聚合的关键。例如可以选择一个查询，比如 countries：India，这将为相同的内容创建一个 bucket，并将文档与搜索查询匹配。
- **Significant terms**：此聚合大部分用于寻找数据中不常见的词项，它比较了索引中包含数据本身的前景集合和包含数据索引的背景集合。此聚合提供前景和背景集合中发生的显著改变的结果。
- **GeoHash**：此聚合用于包含 geo_point 值的字段，它可以动态创建 bucket。它将匹配 geo_points 的文档分组为 bucket。只有瓦片地图（TileMap）的可视化图表中才可以出现 GeoHash。所有的 buckets 聚合在 Kibana 中的显示如下图所示：



4.7.1.2 metric 聚合

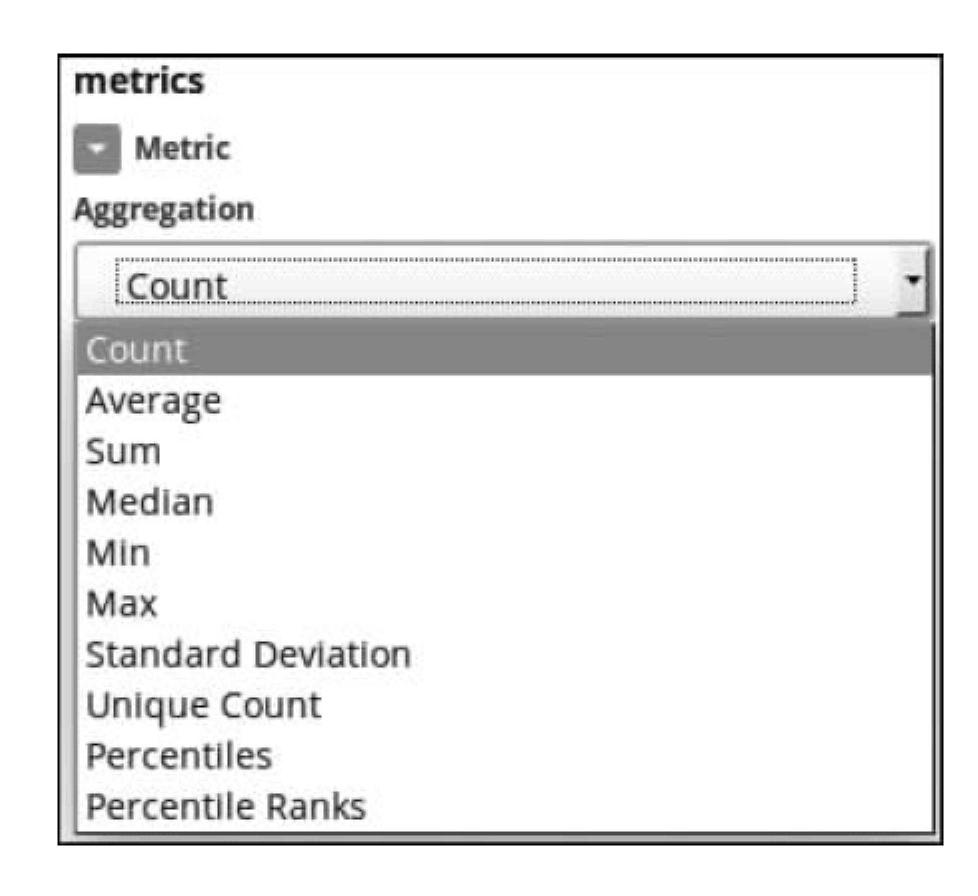
metric（指标）聚合可以执行 bucket 聚合中文档相关指标的计算。该聚合计算诸如计数、平均值、求和、中值、最小值、最大值、标准偏差、去重计数、百分比和百分等级等字段的指标，在 bucket 聚合之上使用。将 bucket 与文档进行聚合之后，可以计算出一些指标，例如文档的数量、所有文档中字段的平均值，以及所有文档中字段的最小值或最大值，以提供每个 bucket 的单一值结果。在图表的可视化中，这些指标可以表示面积图、折线图或直方图

的 Y 轴。

Kibana 中不同的 metric 聚合如下所示：

- **Count(计数)**：此聚合主要用于计算每个 bucket 中的文档数量，并将其返回值作为自己的值。这个值可以用于从文档里显示的任何字段中提取计数。
  例如：为了了解访问网站的用户数量，可以使用站点名称字段进行 bucket 聚合，并使用 metric 聚合来计算该网站的用户数量。
- **Average(平均值)**：此聚合用于计算存储在 bucket 的文档中数字字段的平均值。
- **Sum(求和)**：此聚合用于计算存储在 bucket 的文档中数字字段的总和。
- **Median(中间值)**：此聚合用于计算中间值，它将较高的一半数值与较低的一半数值分隔开。它也可以被视为第 50 个百分点。
- **Min(最小值)**：此聚合用于计算存储在 bucket 的文档中数字字段的最小值。
- **Max(最大值)**：此聚合用于计算存储在 bucket 的文档中数字字段的最大值。
- **Standard Deviation(标准差)**：此聚合用于计算 bucket 中的文档所存储数值的分散情况，它量化了字段值之间的差异。
- **Unique Count(唯一值)**：此聚合用于计算存储在 bucket 的文档中一个字段的唯一值的数目。它的功能类似于 SQL 中的 COUNT(DISTINCT fieldname)。
- **Percentile(百分比)**：此聚合用于计算存储在 bucket 的文档中数字字段的百分比。因为它存储每个 bucket 的多个值，所以属于多值测度聚合的范畴。百分比的结果是指定文档的百分比的值。
- **Percentile Ranks(百分等级)**：此聚合用于计算 bucket 中的文档包含数字字段的百分等级。因为每个 bucket 中存储了多个值，所以它属于多值 metric 聚合的范畴。它指定了 bucket 中值的百分比。

所有的 metrics 聚合在 Kibana 中的显示如下图所示：



了解了用于创建可视化的底层聚合之后，我们来看一看如何构建由 Kibana 提供的多种可视化效果。

4.7.2 可视化画布

可视化画布(Visualization Canvas)由三个主要部分组成：

- **工具栏(Toolbar)**：提供许多选项，比如新建可视化(New Visualization)、保存可视化(Save Visualization)、打开已有的可视化(Open Saved Visualization)、分享可视化(Share Visualization)以及刷新(Refresh)。
- **聚合设计器(Aggregation Designer)**：由两个选项卡组成：数据(Data)和选项(Options)。Data 选项卡用于在创建可视化的基础上定义 metric 和 bucket 聚合。Options 选项卡用于定义为可视化提供的附加选项，这些选项可以根据需要来定义。
- **预览可视化(Preview Visualization)**：用于预览使用 Aggregation Designer 创建的可视化内容。它提供了一个单独的视图页面来自定义、编辑或者修改可视化并立即得到新的视图。

现在，我们来浏览一下可视化的类型。

4.7.3 面积图

面积图(area chart)用于显示数据(文档)在一段时间内的分布，适用于创建堆叠的时间线。它主要用于显示随时间变化的趋势，该区域用各种颜色来描述数据。

4.7.4 数据表

数据表(data table)用于以表格格式显示聚合数据，以文本的形式提供结果。它用于显示文档中存储的任何类型的数据，使之获得更好的理解。

> ℹ数据表可以以原始的或特定的格式导出数据。

4.7.5 折线图

折线图(line chart)用于以行的形式显示一段时间内的数据(文档)分布。它有助于创建时间序列图，主要用于显示随时间变化的趋势或者比较多个时间序列图。

4.7.6 气泡图

气泡图(bubble chart)是折线图的扩展，而不是线，其中每个数据点都显示为一个气泡。通过在 Y 轴上添加其他指标并将指标类型设置为点的大小，可以将折线图扩展/转换为气

泡图。

4.7.7 Markdown 部件

该部件用于在基于文本的输入中显示指令、信息或者解释。它是一个 GitHub 特点的标记，可以显示任何文本、链接、表或代码块，对于与面板有关的信息非常有用。

4.7.8 Metric

Metric 属于数值类统计，用于显示单个数字的所有分析。它显示了基于数据的测度聚合的数目，并且不涉及任何 bucket 聚合。

4.7.9 饼图

饼图(pie chart)用于显示所有数量的比例。它用片(slice)的方式来表示，每一个环整体都由若干部分拼成，这些部分都有各自在整体中的占比。大多数情况下，只有显示比较少的片时才会使用这个可视化效果。

4.7.10 标签云

标签云(tag cloud)用于给数据提供可视化的表示。它通过字体颜色和字体大小来表示一个单词的重要性。metric 聚合的类型决定了字体大小。如果使用了计数，那么出现次数最多的单词字体最大，出现次数最少的单词字体最小。

4.7.11 瓦片地图

瓦片地图(tile map)用于显示基于地理坐标的地理位置，坐标由经纬度决定。它使用 GeoHash bucket 聚合来映射指向其位置的点。

4.7.12 时间序列

时间序列(time series)使用 Timelion 表达式语言来创建基于时间的图表，它有助于推导统计数据、移动平均值以及一系列其他数学函数。可以在本章 4.9 节学习更多有关可视化的内容。

4.7.13 直方图

直方图(vertical bar chart)是用于显示各种数据的通用图表，用于基于和非基于时间的数据。它使用竖条来表示数据，其大小由它们的值决定。条形越长，其值越高；条形越短，其值越小。

下面是 Kibana 为不同的可视化提供的各种**数据**(Data)和**选项**(Options)的列表。

Data 选项可视化如下表所示：

可视化类型	metric	bucket
面积图	Y 轴	X 轴、分割区域、分割图
数据表	Metric	分割行、分割表
折线图	Y 轴	X 轴、分割线、分割区域
Metric	Metric	—
饼图	片大小	分割片、分割图
标签云	标签大小	标签
瓦片地图	值	地理坐标
直方图	Y 轴	X 轴、分割条、分割区域

Option 选项可视化如下表所示：

可视化类型	视 图 选 项
面积图	图表模式(堆叠、重叠、百分比、摆动、轮廓)、平滑线、设置 Y 轴的区段，将 Y 轴刻度与数据对应、按降序顺序排列、图例位置、显示工具提示
数据表	每个页面、显示每个 bucket 或级别的指标、显示部分行、计算每个 bucket 或级别的指标、显示总数、总计
折线图	Y 轴刻度(线性、对数、平方根)、平滑线、显示连接线、显示圆、设置 Y 轴的区段、刻度 Y 轴数据范围、按降序顺序排列、图例位置、显示工具提示
Metric	字体大小
饼图	环形饼图、图例位置、显示工具提示
标签云	文本规模(线性、对数、平方根)、定向(单一、直角、多个)、字体大小、显示标签
瓦片地图	地图类型(缩放圆形标记、阴影圆标记、阴影网格、热点图)、图例位置、显示工具提示、Desaturate Map Tiles、WMS 兼容地图服务器
直方图	条形模式(堆叠、百分比、分组)、Y 轴刻度(线性、对数、平方根)、设置 Y 轴的区段、刻度 Y 轴数据范围、按降序顺序排列、图例位置、显示工具提示

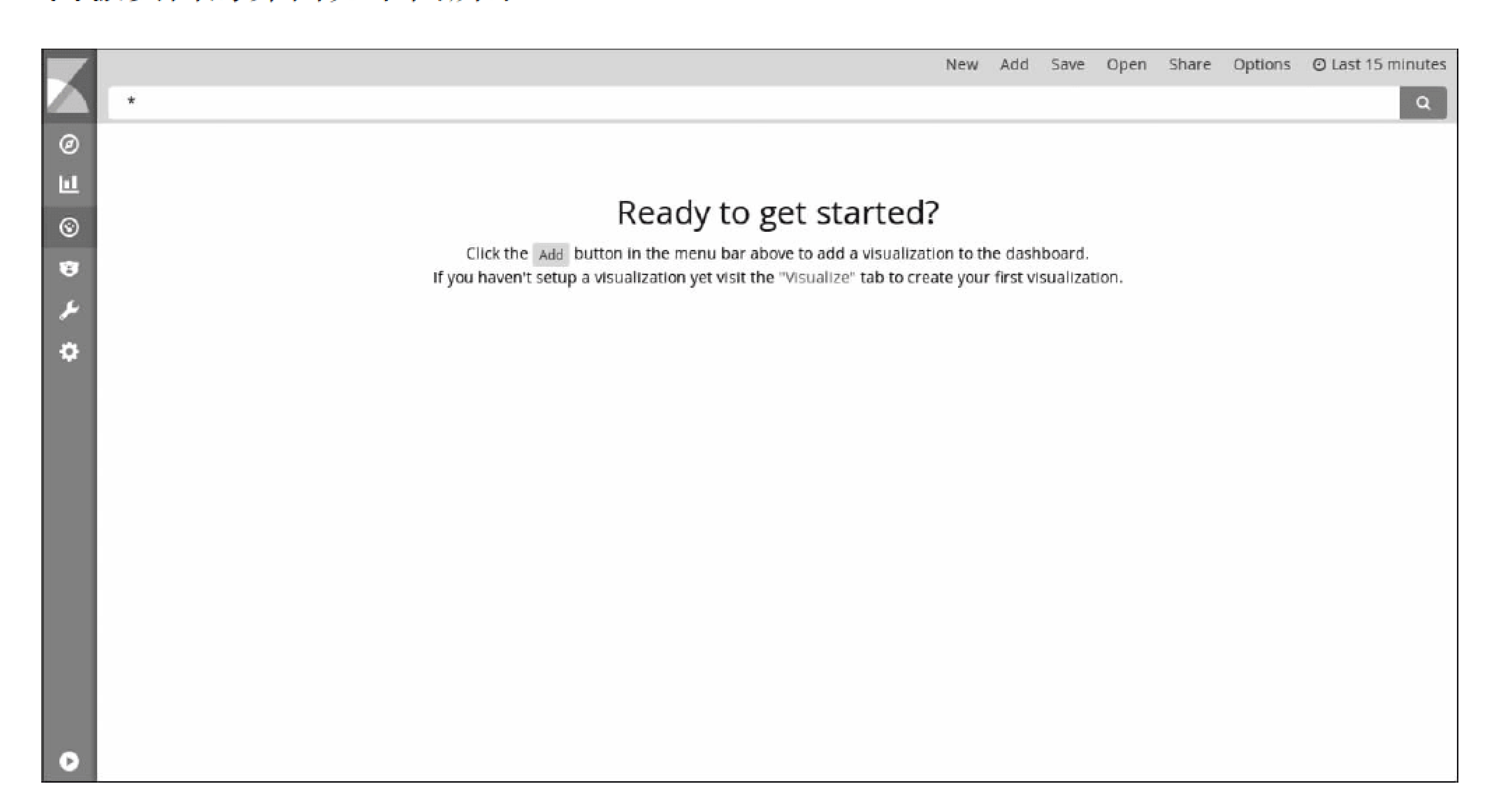
4.8　探索 Dashboard 界面

面板(Dashboard)提供了一个统一的视图，即在一个地方显示所有的可视化效果。面板中提供了多种可视化效果或以任何方式排列的搜索结果集合，它有调整、移动、编辑或者

删除面板上任何可视化内容的功能。当所有可视化效果都在面板中实时更新时，面板为流数据提供实时视图，对可视化内容的更新可以在面板上立即显示出来。面板还提供了使用搜索查询的能力，可以根据搜索结果更新面板中的可视化效果。

> 创建面板时需要事先保存可视化内容或者搜索结果。

面板页面的界面如下图所示：



面板界面非常容易理解，因为它展示了 Kibana 首部的主要部分。它在所有包含时间过滤器的界面中都是相同的。

工具栏由一些选项组成，比如：

- **搜索栏（Search Bar）**：查询面板，类似于**查询**页面的查询功能。
- **新建面板（New Dashboard）**：创建一个新的面板。

> 如果已经创建了面板，并已经向其中添加了可视化效果，但是没有保存面板就单击了新建面板，之后未保存的面板将被清空，并且其中添加的可视化内容也会消失。

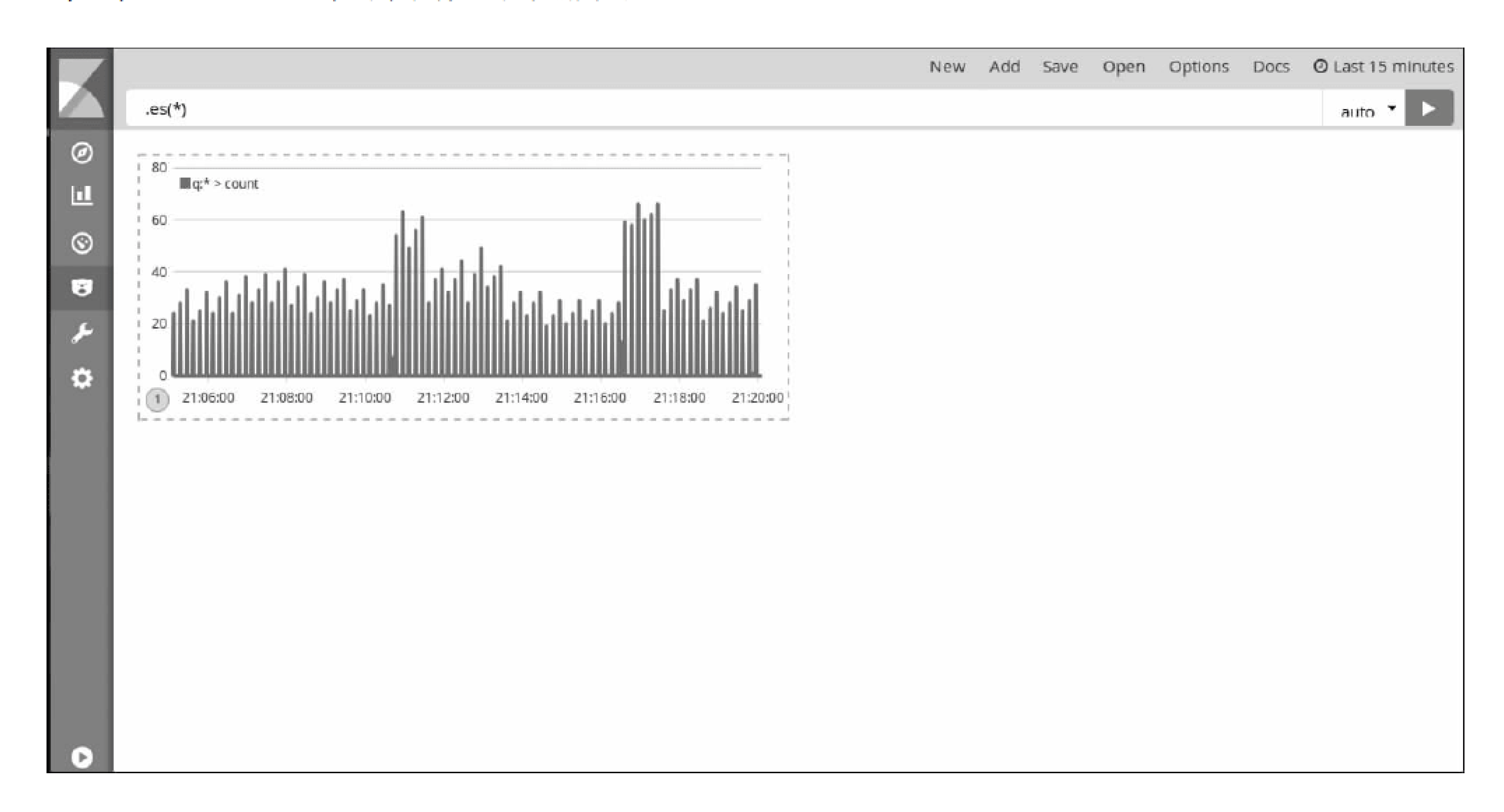
- **添加（Add）**：添加已保存的可视化和搜索面板。
- **保存面板（Save Dashboard）**：保存包含可视化的面板。
- **打开面板（Open Dashboard）**：加载已保存的包含可视化的面板。
- **共享（Share）**：通过共享链接或者将链接嵌入到 Web 页面中来共享面板，此外，还允许生成一个简短的 URL 来共享链接。
- **选项（Options）**：为面板选择主题，可选择深色主题或者亮色主题。

4.9 了解 Timelion

Timelion 是随着引入 Elastic Stack 的 Kibana UI 中新增的内容，用于分析和可视化时间序列数据。它提供将多个数据源合并成一个可视化统计图表的能力，并给出了一系列可以使用的数学计算，例如累加、求导和求平均数等。

Timelion 位于 Kibana UI 左边的导航栏中，在**面板**和**开发工具**图标之间。它有自己的语言和解释，使我们很难上手开始应用。不过，它有一个很好的内置文档和教程可指导我们如何开始使用 Timelion。

单击 Timelion 时，界面如下图所示：



在上面的截图中可以看到，输入框中已经添加了一个默认的表达式.es(*)，这意味着它将在所有的索引中查询 Elasticsearch 中的全部数据。图中的内容表示 Elasticsearch 中的文档数。

将表达式从.es(*)修改为.es(index=logs)可查询某个特定的索引，例如日志。之后，它将显示在 Elasticsearch 中 logs 索引中的文档数。

下面，简单地了解一下 Timelion 的界面：

- **新建(New)**：为创建可视化而开启一个新的 Timelion 页面。
- **添加(Add)**：在同一 Timelion 页面内添加图表。
- **保存(Save)**：保存 Timelion 页面。它提供两个选项，可保存 Timelion 表或将当前表达式保存为 Kibana 面板。
- **打开(Open)**：打开已保存的 Timelion 图表。

- **选项(Options)**：提供了指定行数和列数的选项。
- **文档(Docs)**：提供了 Timelion 入门的文档。

其他选项：

- **时间过滤器(Time Filter)**：提供了时间过滤选项来过滤数据。
- **搜索框(Search Box)**：提供了使用语言和表达式来分析数据的用法。
- **时间间隔(Time Interval)**：指定要定义的时间间隔。如果设置为自动(auto)，则根据时间过滤器中的时间来选择时间间隔。它就在搜索框旁边，可以设置为 1 秒、1 分钟、1 天、1 年或自定义的时间间隔。

要开始学习如何使用 Timelion，先单击 Timefilter 选项旁边工具栏中的 **Docs** 选项。单击后，将会出现函数写法的参考信息，该信息描述了 Timelion 中的所有可用函数。该界面如下图所示：

New Add Save Open Options Docs Last 15 minutes

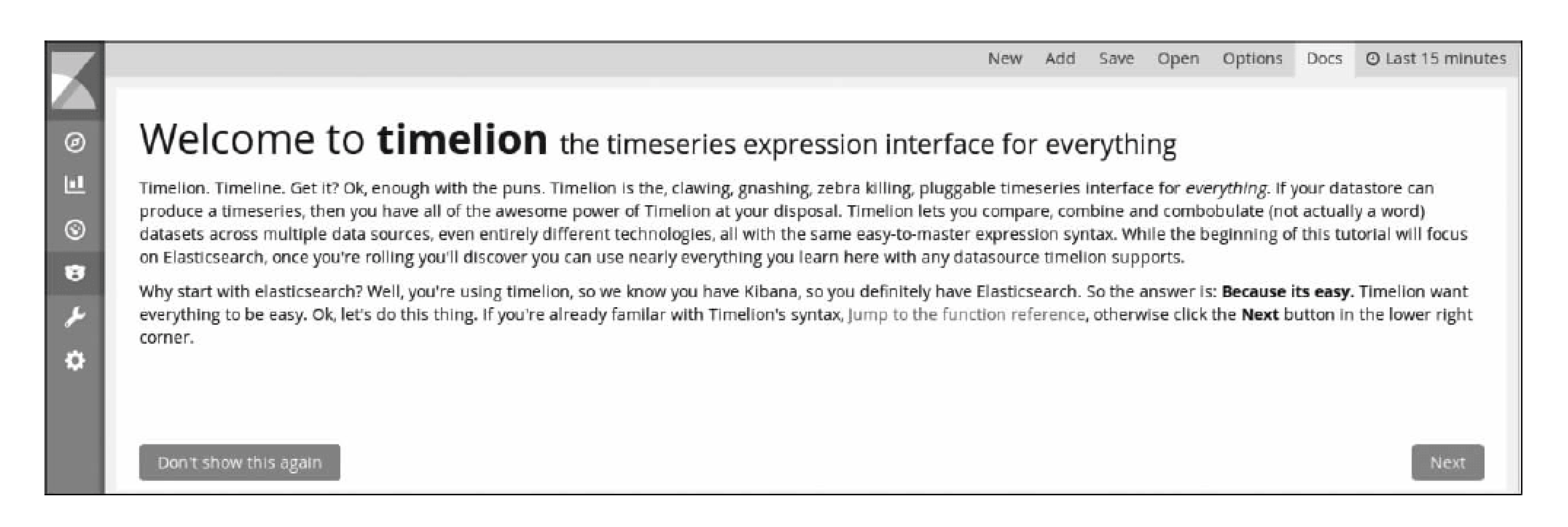
Function reference

Click a function for details and arguments or return to the tutorial.

.abs()	Return the absolute value of each value in the series list
.add()	Adds the values of one or more series in a seriesList to each position, in each series, of the input seriesList
.bars()	Show the seriesList as bars
.color()	Change the color of the series
.condition()	Compares each point to a number, or the same point in another series using an operator, then sets its valueto the result if the condition proves true, with an optional else.
.cusum()	Return the cumulative sum of a series, starting at a base.
.derivative()	Plot the change in values over time.

可以向下拉滚动条查看更多功能，这里有将近 50 个功能。它还提供了从诸如 Quandl 和世界银行等第三方站点获取数据的功能，这些数据以 API 调用的形式提供。

同样，从之前的截图中，还可以看到有一个教程的参考资料，其中有对 Timelion 的更深入的理解。若要浏览本教程，请单击**返回教程(return to the tutorial)**的链接，之后界面会跳转到**欢迎访问 Timelion(Welcome to Timelion)**页面，如下图所示：



可以单击 **Next** 按钮来学习关于 Timelion 的知识，该按钮在教程页面的右下角。

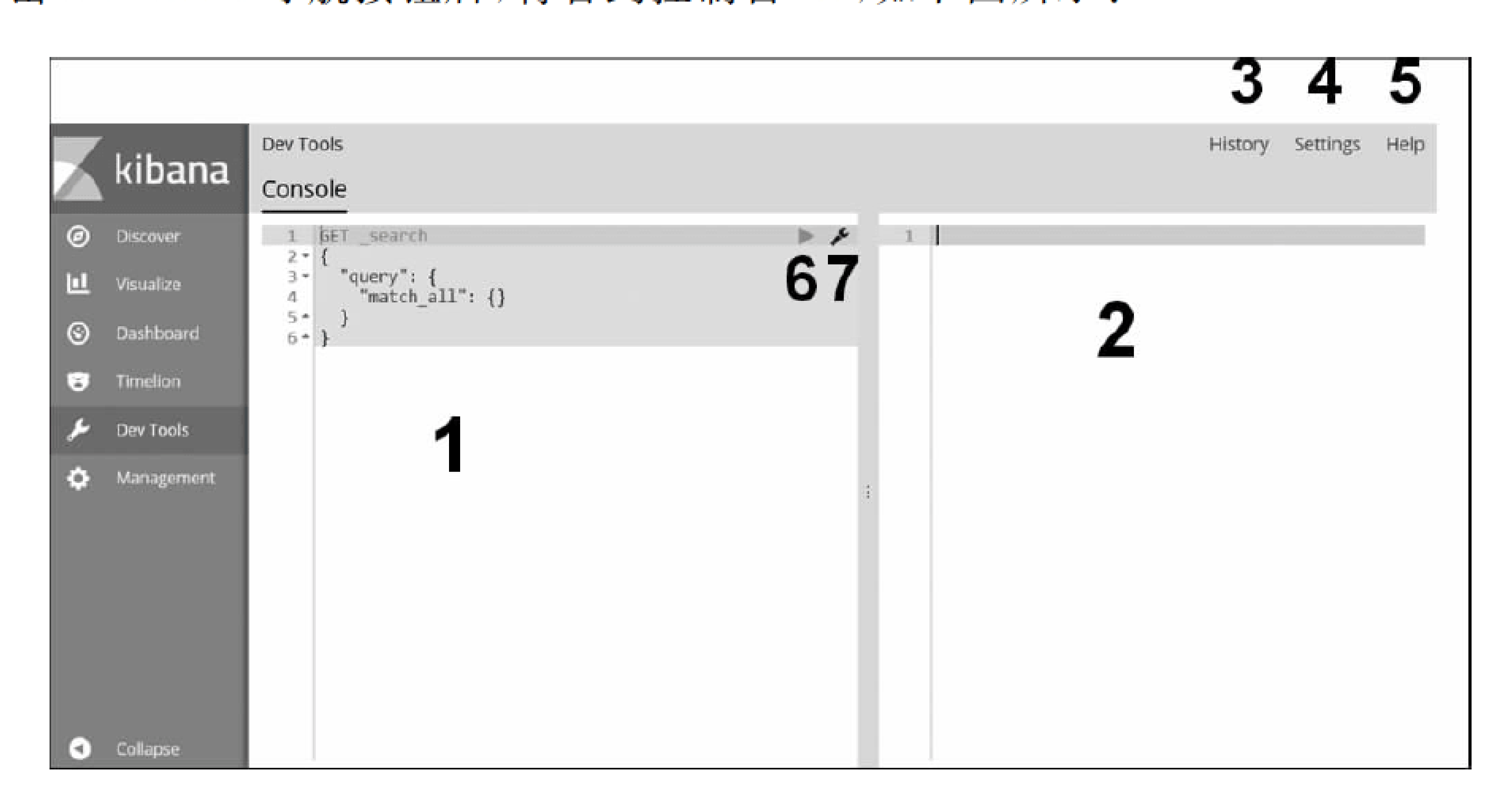
让我们看一下在添加图表后出现的几个选项：

- **移除（Remove）**：从 Timelion 表中移除图表。
- **排序（Drag to Order）**：重新排列各种图表的位置。
- **全屏（Full Screen）**：全屏显示已创建的可视化内容。

4.10　探索开发者工具

开发者工具（Dev Tools）是帮助开发人员的开发工具。在 Kibana 中，它用于控制台 UI。它提供了一个简洁的用户接口，可以使用由 Elasticsearch 客户端开放的 REST API 来访问 API 查询。控制台允许我们从 Web 浏览器中调用任何 API。它的接口提供了一种简洁的方法来调用和生成 JSON 格式，从而可以简洁的方式查看结果。它在 Elasticsearch 集群的 HTTP 层上运行。

单击 **Dev Tools** 导航按钮后，将看到控制台 UI，如下图所示：



为了更好地理解上图中用标号 1～7 所标示的所有功能，描述如下：

（1）编辑器窗格（Editor Pane）：在这个区域里写需要运行的命令。它使用类似于 cURL 的语法，并以一种简单的方式呈现出来。例如默认情况下，一个查询语句在控制台中如下所示：

```
GET _search
{
    "query": {
        "match_all": {}
    }
}
```

它可以被解释为以下的 Elasticsearch 查询：

```
curl -XGET "http://localhost:9200/_search" -d'
{
    "query": {
        "match_all": {}
    }
}'
```

GET 是要使用的访问方式，_search 是搜索在 Elasticsearch 中的所有索引的端点。该区域可为 API、索引、类型等提供自动完成和自动建议的内容。可以粘贴完整的 cURL 命令，它们将自动转换为控制台所采用的语法。

> 控制台将查询 Kibana 连接到 Elasticsearch 的主机地址。默认情况下，该地址为 localhost:9200。

（2）响应窗格（Response Pane）：无论我们请求什么，控制台都会向 Elasticsearch 发送请求，并且返回的结果将在这个区域中以 JSON 格式显示出来。

（3）这个选项展示请求历史，它存储了最近的 500 条请求历史。

（4）通过此选项，可以更改字体属性，自动完成设置等。

（5）这是一个为用户提供的帮助选项，展示了一个简单的请求，并且列出了所有的快捷键。

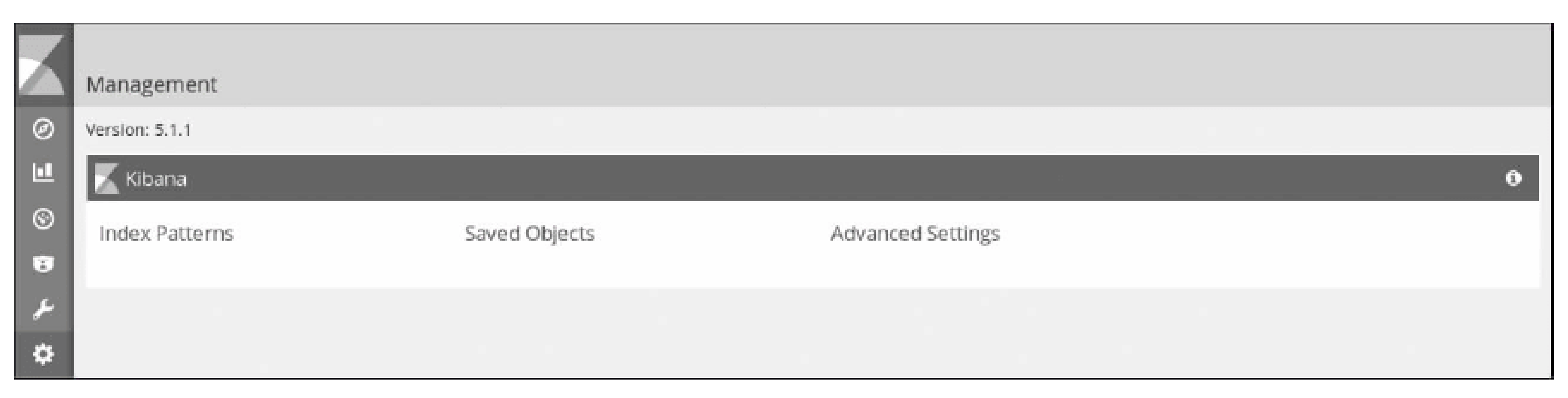
（6）无论输入什么请求，都可以通过单击绿色运行按钮来运行该请求。

（7）这个扳手图标提供了一个选项，可以将请求复制为 cURL 命令。如果文本应该自动缩进，则单击该图标即可完成。如果请求信息已经被正确格式化，并且选择了 auto indent，那么控制台将会将完整的请求合并成一行，这对于调试是很有用的。

> 可以在编辑窗格中编写多个请求，这些请求将按照指定的顺序逐一执行。

4.11 探索设置界面

设置（Management）界面提供了自定义和调整各种 Kibana 相关属性的方法。这个页面包含多类选项，用来理解每个选项涉及的各种设置。各种选项如下图所示：



下面我们来深入学习选项中提供的各种设置。

4.11.1 索引模式

该选项提供了配置索引或索引模式的功能，所配置的索引或索引模式可以与 Kibana 一起使用。Kibana 中提供了许多索引模式，这些索引模式已经被配置为与 Kibana 一起使用，并且能够查看在索引中显示的各种字段的相关信息。

可以配置许多索引类型，比如：

- **索引名称（Index Name）**；
- **索引模式（Index Patterns）**。

索引模式被分成如下两种搜索类型：

- **通配符搜索**：logstash-* 将获取所有以 logstash-开头的索引名称。
- **基于事件的时间**：[logstash-]YYYY-MM-DD 将以一个模式来获取所有的索引，其中索引名称以 logstash-开头，之后是事件/日期时间。

对于索引的额外设置，单击索引可以查看更多信息，比如：

- 列出索引中所有字段；
- 字段的属性，如类型、格式、可搜索、聚合、分析、排除和控制；
- 将索引设置为默认索引的选项；
- 更新索引里的字段列表；
- 从 Kibana 中移除索引模式；
- 管理字段属性，比如更改字段格式。

单击索引名称之后，可以在字段之外获取其他设置，比如：

- Scripted fields：创建动态生成的字段以计算信息。
- Source filters：排除来自_source 字段的字段。因为_source 字段包含所有的字段，其中一些字段可能不重要。因此，可以直接从_source 字段移除这些不重要字段。

4.11.2 已保存的对象

这个选项用于管理多个已保存的对象，比如已保存的搜索结果、可视化内容和面板。这些已保存对象用于查看、编辑、导出、导入和删除对象。它还提供了将所有保存对象一次性导出为搜索结果、可视化内容或面板的功能。

4.11.3 高级设置

这个选项包含了诸如实验性的、未加入文档的或未支持的高级设置。它提供了额外的功能来调整 Kibana 的设置和控制。它主要用于高级用户的使用，但如果使用不当会造成不

可预料的结果。同样，这里执行的任何删除设置，都将在 Kibana 中永久生效。

下面，看一下这个选项卡提供的部分设置：

- **dateFormat**：改变文档中的日期格式。
  例如：默认设置为：MMMM DD YYYY，HH：mm：ss. SSS(July 27th 2016，12：46：01. 000)
  可以更改为：MMMM DD YYYY(July 27 2016)
- **dateFormat**：**tz**：用于更改指定时区的日期格式。
  例如：默认设置为：Browser-IST(July 27th 2016，12：45：25. 000)
  可以更改为：Asia/Singapore (June 27th 2016，15：15：25. 000)
- **doc_table**：**highlight**：指定是否将结果高亮。
  默认设置为：True (在 **Discover** 界面和**已保存的**搜索结果面板中)
- **history**：**limit**：显示查询输入的最近值。
  默认设置为：10
- **savedObjects**：**perPage**：使用加载对话框，显示每个网页中保存的对象数。
  默认设置为：5
- **timepicker**：**timeDefaults**：设置默认的 Kibana 时间过滤器。
  默认设置为：{"from"："now-15m"，"to"："now"，"mode"："quick"}
  可以更改为：{"from"："now-30m"，"to"："now-15m"，"mode"："relative"}
- **timepicker**：**timeDefaults**：改变默认时间过滤器，并将展示的结果从 30 分钟前更改为 15 分钟前。
- **dashboard**：**defaultDarkTheme**：设置**面板**页面的默认主题。
  默认设置为：false，可更改为 true。面板打开时它将始终显示深色主题的面板。

了解 Kibana 提供的多个选项后，如果仔细观察，就会发现还有一些选项，即提供 Kibana 最少量的信息。单击右上角的信息图标，将提供以下信息：

- 版本号；
- 提交 SHA 散列码。

同样，在**管理**界面标题和 Kibana 设置选项之间也提到了版本号。

有时可能会有一些问题，Kibana 不能启动或者不显示在查询界面中的任何数据。为了知道 Kibana 是否正确启动，可以单击下面的 URL：

http://localhost:5601/status

这里的 localhost 是你的 Kibana 服务器的主机地址。

上面的 URL 会提供一些信息，下面一节会对这些信息进行介绍。

4.11.4 状态

状态用于提供 Kibana 服务器的状态和其他信息，比如：

- 堆总大小；
- 已使用的堆大小；
- 负载；
- 平均响应时间；
- 最大响应时间；
- 每秒请求；
- 关于 Kibana 安装插件的明细及其名称和状态。

4.12 综合应用

学习 Elasticsearch、Logstash 和 Kibana 之后，让我们使用这些组件创建端到端的数据管道(pipeline)，来解析从 Logstash 到 Elasticsearch 的数据，并使用 Kibana 将数据可视化。我们将使用 CSV 文件作为输入数据，分析和创建数据的可视化。这将帮助我们通过联合使用三个组件来快速入门并创建端到端的数据管道 pipeline。学习这一章中的例子时，假定已经按照第 1 章的描述成功安装了 Elasticsearch、Logstash 和 Kibana。

4.12.1 输入数据

下面将使用**美国农业部经济研究局**(**USDA ERS**)提供的输入数据，这是美国各地区居民从 1970 年到 2014 年受教育程度的数据。这些数据提供了人们关于受教育程度的信息，包含未获得高中文凭的人、只有高中文凭的人、只完成了大专(1～3 年)学位的人，还有获得了大学本科四年学位的人等信息。所有这些信息都是从 1970 年到 2014 年，以十年为间隔展示的。它代表了按照国家和地区名称划分的接受教育的人数和比例。这些数据还提供了从 2003 年到 2013 年关于 Rural-Urban Continuum Code(RUCC)和 Urban Influence Code(UIC)的信息。文件格式为.csv。

> 数据来自 http://www.ers.usda.gov/data-products/county-level-data-sets/download-data.aspx。
>
> 关于 UIC 的信息详见 http://www.ers.usda.gov/data-products/urban-influence-codes/documentation.aspx。
>
> 关于 RUCC 的信息详见 http://www.ers.usda.gov/data-products/rural-urban-continuum-code s/documentation.aspx。

这个文件包含大量的头(headers)信息或者说列。我们将使用部分头信息来分析数据和创建可视化图表。

让我们看一下这部分工作流程。所使用的输入数据会被插入到 Logstash 进行处理，输出的数据将存储在 Elasticsearch 中，之后 Kibana 将使用这些数据来分析数据。

具体工作流程如下：

(1) 创建一个 Logstash 配置文件，并执行如下步骤：

① 使用 file 输入插件将数据插入到 Logstash 中。

② 使用 CSV 过滤器来命名文件中的列，并将字段的数据类型从字符串更改为适当的数据类型。

③ 使用 Elasticsearch 输出插件将数据存入 Elasticsearch，也可以使用标准输出的方式将日志信息输出到屏幕上。

(2) 在 Kibana 中创建一个新的索引模式。

(3) 在 Kibana 中分析数据并对数据执行可视化。

4.12.2 创建 Logstash 配置文件

我们已经了解了用于输入、过滤和输出的各种插件。为了便于理解，下面将展示每个输入、过滤和输出插件的配置文件的内容。

通过执行以下代码来使用 Input 插件：

```
input{
    file{
        path =>"/usr/share/logstash/InputData/Education.csv"
        start_position =>"beginning"
    }
}
```

在前面的 Input 插件中，使用了文件输入插件，并指定了读取输入数据的文件路径，并设置了读取数据从开头到结尾的全部内容。通过实现以下代码来使用 Filter 插件：

```
filter
{
  csv{
  columns=>
    [
    "State",
    "AreaName",
    "2003Rural-urbanContinuumCode",
```

```
        "2003UrbanInfluenceCode",
        "2013Rural-urbanContinuumCode",
        "2013UrbanInfluenceCode",
      "Lessthanahighschooldiploma,1970", "Highschooldiplomaonly,1970",
        "Percentofadultswithlessthanahighschooldiploma,1970",
        "Percentofadultswithlessthanahighschooldiploma,2010-2014",
        "Percentofadultswithahighschooldiplomaonly,2010-2014",
        "Percentofadultscompletingsomecollegeorassociate'sdegree,2010-2014",
        "Percentofadultswithabachelor'sdegreeorhigher,2010-2014"
        ]
    }
    mutate {
        convert =>["2003Rural-urbanContinuumCode","integer"]
        convert =>["2003UrbanInfluenceCode","integer"]
        convert =>["2013Rural-urbanContinuumCode","integer"]
        convert =>["2013UrbanInfluenceCode","integer"]
        convert =>["Lessthanahighschooldiploma,1970","integer"]
        convert =>["Highschooldiplomaonly,1970","integer"]
        convert =>
        ["Percentofadultswithlessthanahighschooldiploma,1970","float"]
        convert =>
        ["Percentofadultswithlessthanahighschooldiploma,2010-2014","float"]
        convert =>
        ["Percentofadultswithahighschooldiplomaonly,2010-2014","float"]
        convert =>
    ["Percentofadultscompletingsomecollegeorassociate'sdegree,2010-2014","float"]
        convert =>
    ["Percentofadultswithabachelor'sdegreeorhigher,2010-2014","float"]
        }
    }
```

在前面的 Filter 插件中，我们使用了 CSV 过滤器插件，其中指定了 CSV 中的列名，并使用 mutate 对象将字段的数据类型从字符串改为浮点数或者整数。

> 默认情况下，CSV 中的列名一般命名为 column 1、column 2 等。同样在默认情况下，Logstash 将列解析为字符串类型。

通过实现下面的代码来使用 Output 插件：

```
output{
  elasticsearch {
    hosts =>"localhost:9200"
    index =>"education"
  }
  stdout { codec => rubydebug }
}
```

在上面的 Output 插件中，通过指定主机名、索引名称和文档类型等属性来存储 Elasticsearch 中的数据。同样，使用 stdout 可以将输出数据打印到正在运行的 Logstash shell 命令界面中。

将之前创建的 Logstash 配置以文件名 education.conf 存储在/usr/share/logstash 文件夹下。转到 Logstash 存储目录，并执行下面的命令运行带有配置文件的 Logstash：

```
$ sudo bin/logstash -f education.conf
```

> 确保在运行带有配置文件的 Logstash 之前，Elasticsearch 正在以第 1 章中所描述的状态运行。

Logstash 将运行和解析每一种输入，并将这些输入数据存储在 Elasticsearch 的 education 索引中。控制台中的日志显示了输入的数据如何存储在 Elasticsearch 中。输出如下：

```
Settings: Default pipeline workers: 4
Logstash startup completed
{ "message" =>"US,UnitedStates,,,,,"523,73,312","341,58,051",47.7,
13.7,28.0,29.1,29.3
"@version"=>"1",
"@timestamp" =>"2017-02-16T05:02:31.878Z",
"path" =>/usr/share/logstash/InputData/Education.csv
"host" =>"ubuntu",
"State" =>"US",
"AreaName" =>"UnitedStates",
"2003Rural-urbanContinuumCode" =>nil,
"2003UrbanInfluenceCode" =>nil,
"2013Rural-urbanContinuumCode" =>nil,
"2013UrbanInfluenceCode" =>nil,
"Lessthanahighschooldiploma,1970" =>"523,73,312",
"Highschooldiplomaonly,1970" =>"341,58,051",
```

```
"Percentofadultswithlessthanahighschooldiploma,1970" =>"47.7",
"Percentofadultswithlessthanahighschooldiploma,2010-2014" =>"13.7",
"Percentofadultswithahighschooldiplomaonly,2010-2014" =>"28.0",
"Percentofadultscompletingsomecollegeorassociate'sdegree,2010-2014" =>
"29.1",
"Percentofadultswithabachelor'sdegreeorhigher,2010-2014" =>"29.3"
}
```

4.12.3 使用 Kibana

将数据从 Logstash 送入 Elasticsearch 后，让我们来看看存储到 Elasticsearch 中的数据。可以按照第 1 章中介绍的方式运行 Kibana，也可以将其作为服务来运行，或者运行 bat 文件(限 Windows 用户)。

下面的命令将 Kibana 终端作为服务启动：

```
sudo service kibana start
```

或者，转到 Kibana 安装目录后，使用以下命令启动 Kibana(限 Ubuntu 系统)：

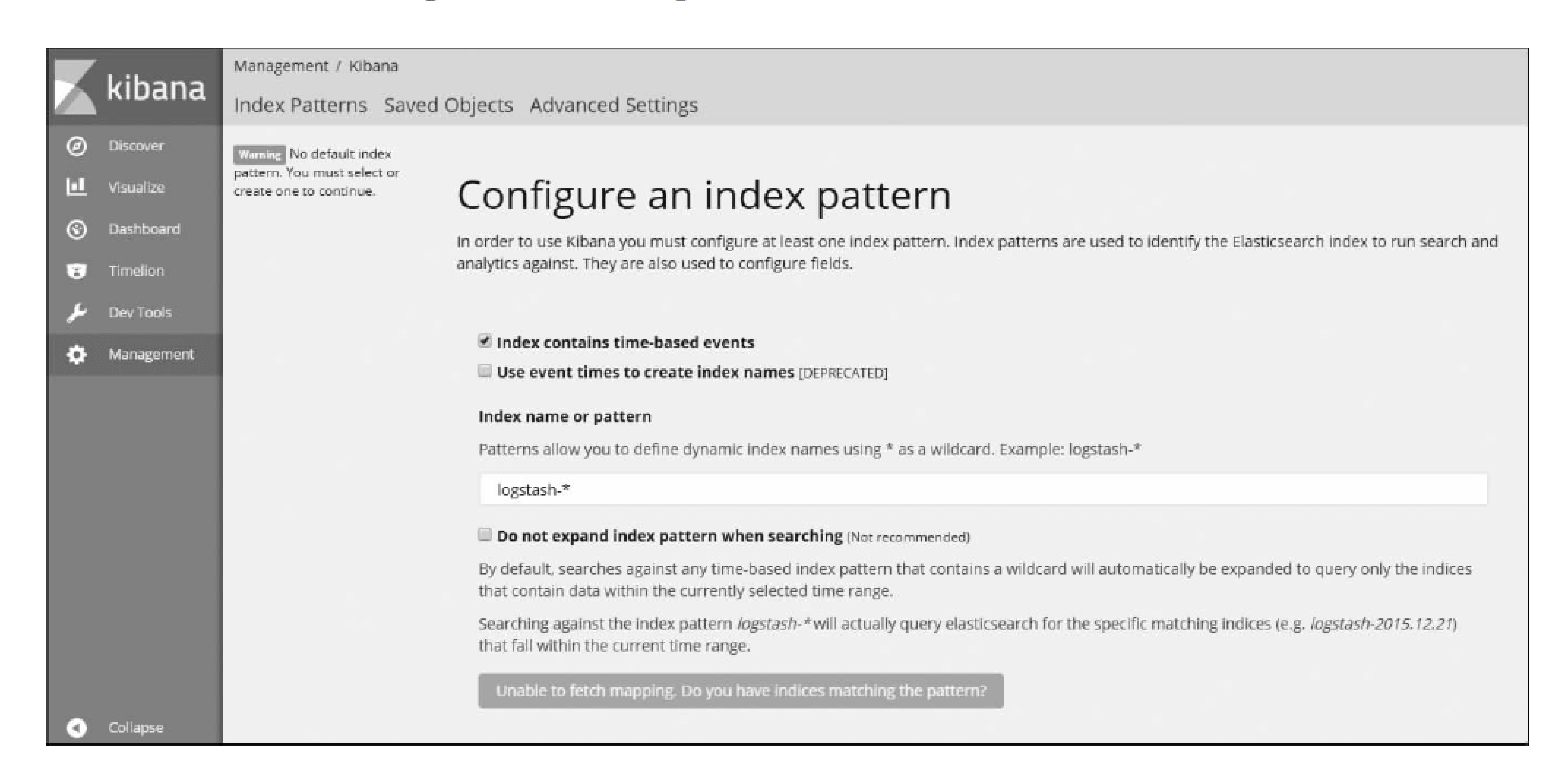
```
bin/kibana
```

或者，在 Windows 系统中直接单击 Kibana 安装目录中的 Kibana.bat 文件来运行 Kibana。

在浏览器中输入以下 URL 验证 Kibana 是否已经启动：

```
http://localhost:5601
```

配置索引模式(Configure an index pattern)的界面如下图所示。



可以通过指定索引名称/模式 education 配置索引模式。此外，取消勾选 **Index contains time-based events** 选项，因为数据中没有基于时间的相关字段。最后，单击 Create 按钮来配置索引。

单击 **Create** 按钮后，我们来验证一下数据是否被索引，以及它们的字段名和数据类型是否正确。转到 **Management** 页面，单击 education 索引名称，该索引名称在左侧 **Index Patterns** 列表中。之后界面中将列出索引中的字段并提供诸如字段名、数据类型、格式以及字段是否可搜索、聚合、分析或排除的细节信息。接着转到 **Discover** 界面来查询数据。单击 Discover 导航按钮，并在左侧窗格找到这个索引名称。如果索引名称是 education，那么就可以在这个页面中查询数据，或者通过单击索引名称旁边的箭头按钮来选择 education 的索引名称。

Discover 页面的数据如下图所示。



通过查看与每个字段相关联的数据可了解这些文档，以便更好地获知这些字段所表达的内容以及字段中包含的值。可以找到诸如 State 和 AreaName 等字段中的值，单击可用字段(Available Fields)来查看该字段中最常见的值。为想查看的数据添加字段，可以很容易地分析这些数据。

分析数据后，我们来执行可视化。创建并保存可视化内容，这些内容可以用来创建一个面板。下面为以下场景构建可视化：

- 基于 2003 年 RUCC 统计的前几个国家；
- 基于 2003 年 UIC 统计的前几个国家；
- 1970 年学历低于高中毕业水平的前五个地区；

- 1970 年学历达到高中毕业水平的前五个地区；
- 1970 年按地区和州统计的低于高中毕业文凭的成人比例；
- 根据数量统计和 2013 年 RUCC 统计的前几个州；
- 包含总记录、2013 年 RUCC 总数、2013 年 UIC 总数、州总数、地区名称总数、未获得高中文凭(2010—2014 年)的成年人平均百分比、仅有高中文凭的成年人的平均百分比、有大学或者大专学历的成年人的平均百分比、有学士学位或者更高学位的成年人的平均百分比等数据的总览。

4.12.3.1 基于 2003 年 RUCC 的前几个州

在这个场景中，希望创建一个条形图来展示前五个 RUCC 值以及州的数量统计。

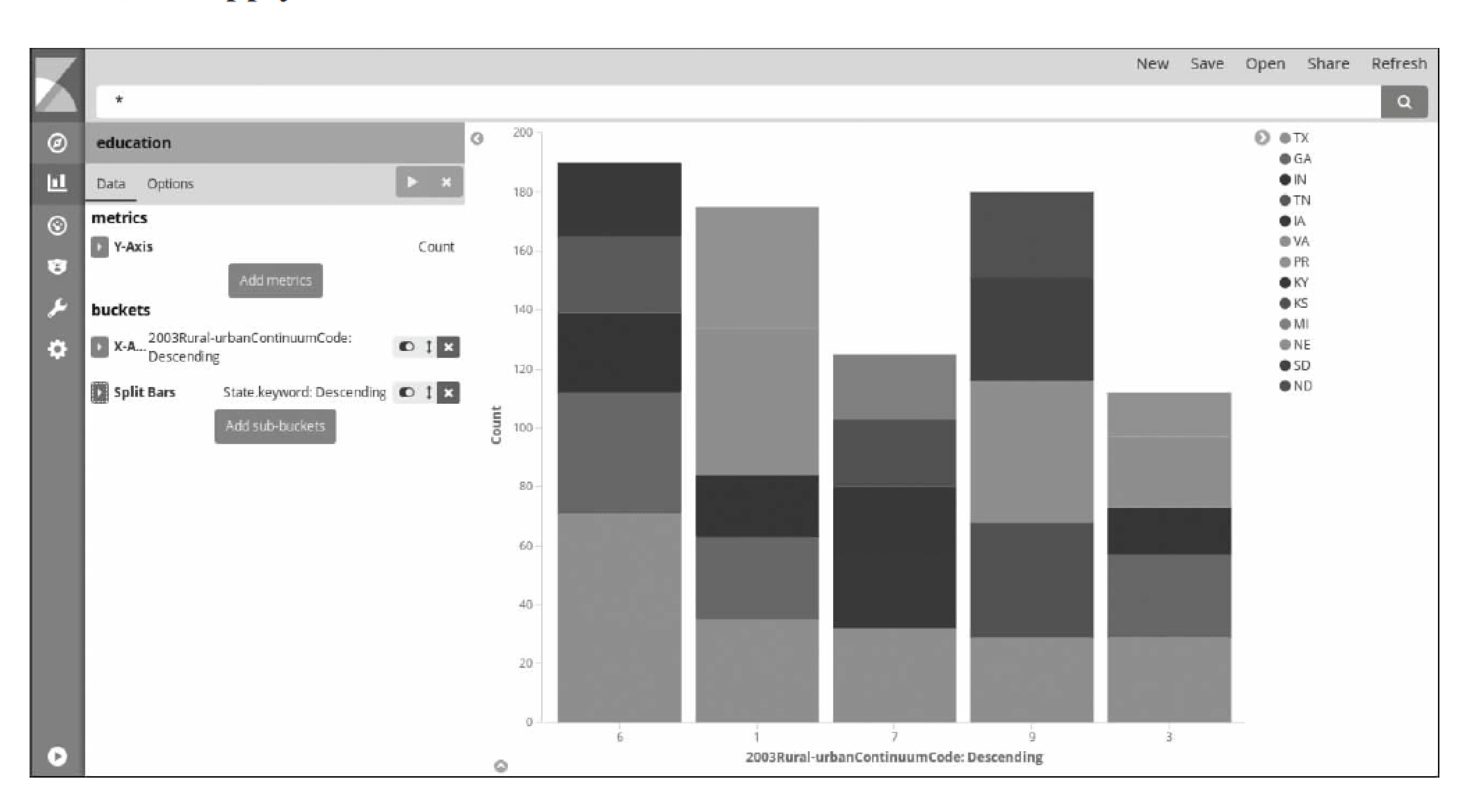
(1) 单击 **Visualize** 选项卡并且选择直方图(Vertical bar chart)来创建一个新的可视化内容。

(2) 新建搜索并选择一个搜索源，指定索引名称为 education(在 Kibana 中配置多个索引的情况下)。

(3) 在 metrics 部分，将 Y 轴的聚合方式设置为 **Count**。在 buckets 聚合中，设置 X 轴的聚合方式为 Terms，并应用在 2003Rural-urbanContinuumCode 字段上。该字段以 metric:Count 为依据来执行降序排序，并将 size 设置为 5。

(4) 在子聚合中，将 **Split Bars** 的聚合方式设置为 Terms，并应用在 State. keyword 字段上。该字段以 metric:Count 为依据降序排序，并将其 size 设置为 5。

(5) 单击 **Apply** 按钮查看可视化结果，如下图所示。



将这个可视化统计图以 TopStatesBasedon2003RUCC 名称保存，这将用于创建面板。

4.12.3.2 基于 2003 年 UIC 的前几个州

在这个场景中，希望创建一个条形图来展示 UIC 统计数量最多的前五个州，并基于这些州的统计，展示前三种 2003 年 UIC 的值。

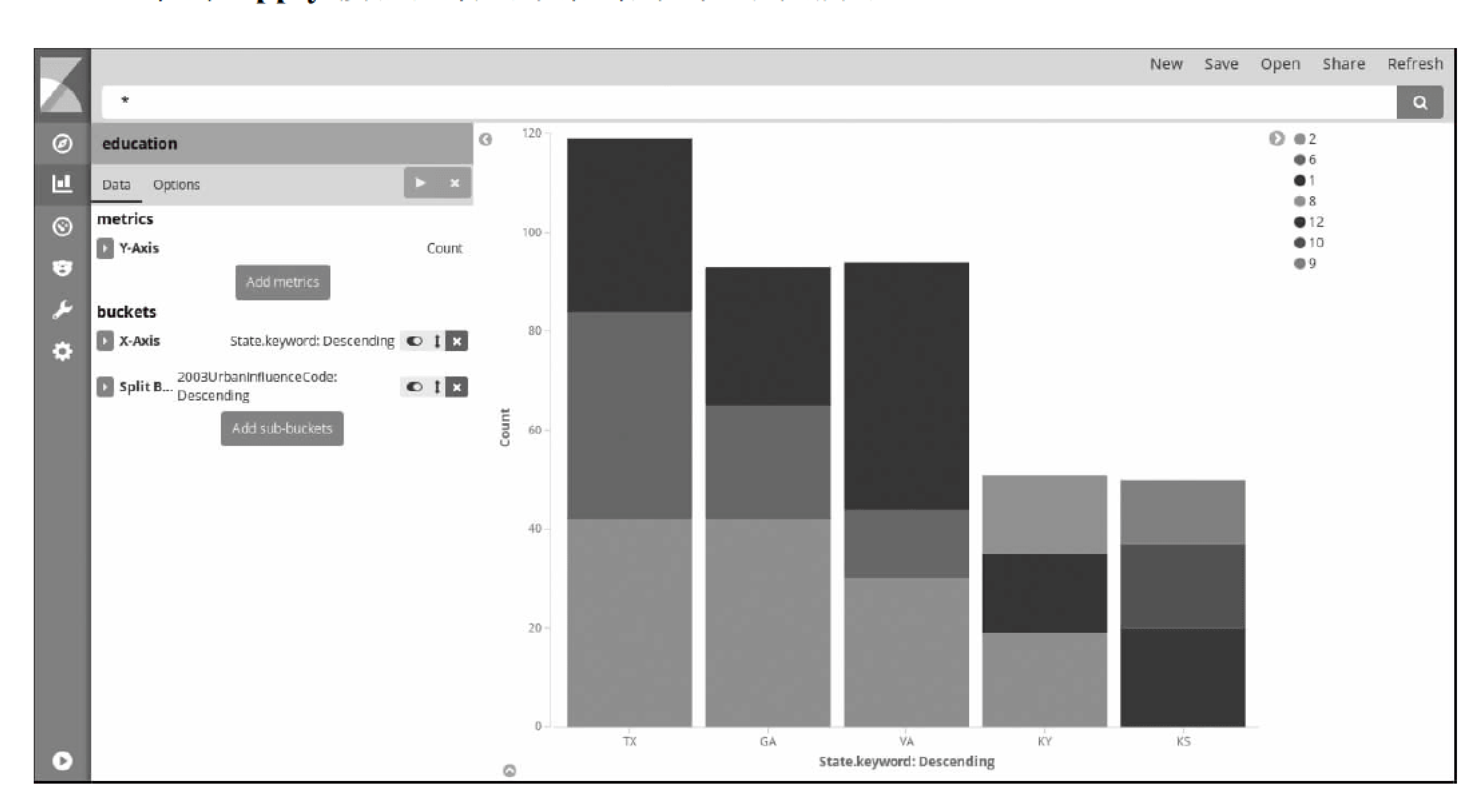
（1）单击 **Visualize** 选项卡，并且选择直方图来创建一个新的可视化内容。

（2）新建搜索并选择一个搜索源，指定索引名称为 education（在 Kibana 中配置多个索引的情况下）。

（3）在 **metrics** 部分，将 **Y 轴**的聚合方式设置为 **Count**。在 **buckets** 聚合中，设置 **X 轴**的聚合方式为 Terms，并应用在 State.keyword 字段上。该字段以 metric：Count 为排序依据，执行降序排序，并将 size 设置为 5。

（4）在子聚合中设置 **Split Bars** 的聚合方式为 Terms，并应用在 2003UrbanInfluenceCode 字段上。此处以 metric：Count 为依据降序排序，并将其 size 设置为 5。

（5）单击 **Apply** 按钮查看可视化结果，如下图所示。



将这个可视化统计图以 TopStatesBasedon2003UIC 名称保存，这将用于创建面板。

4.12.3.3 1970 年学历低于高中毕业水平的前五个地区

在这个场景中，希望创建一个条形图，展示 LessthanHighSchoolDiploma1970 字段的前五个值及其对应的地区名称：

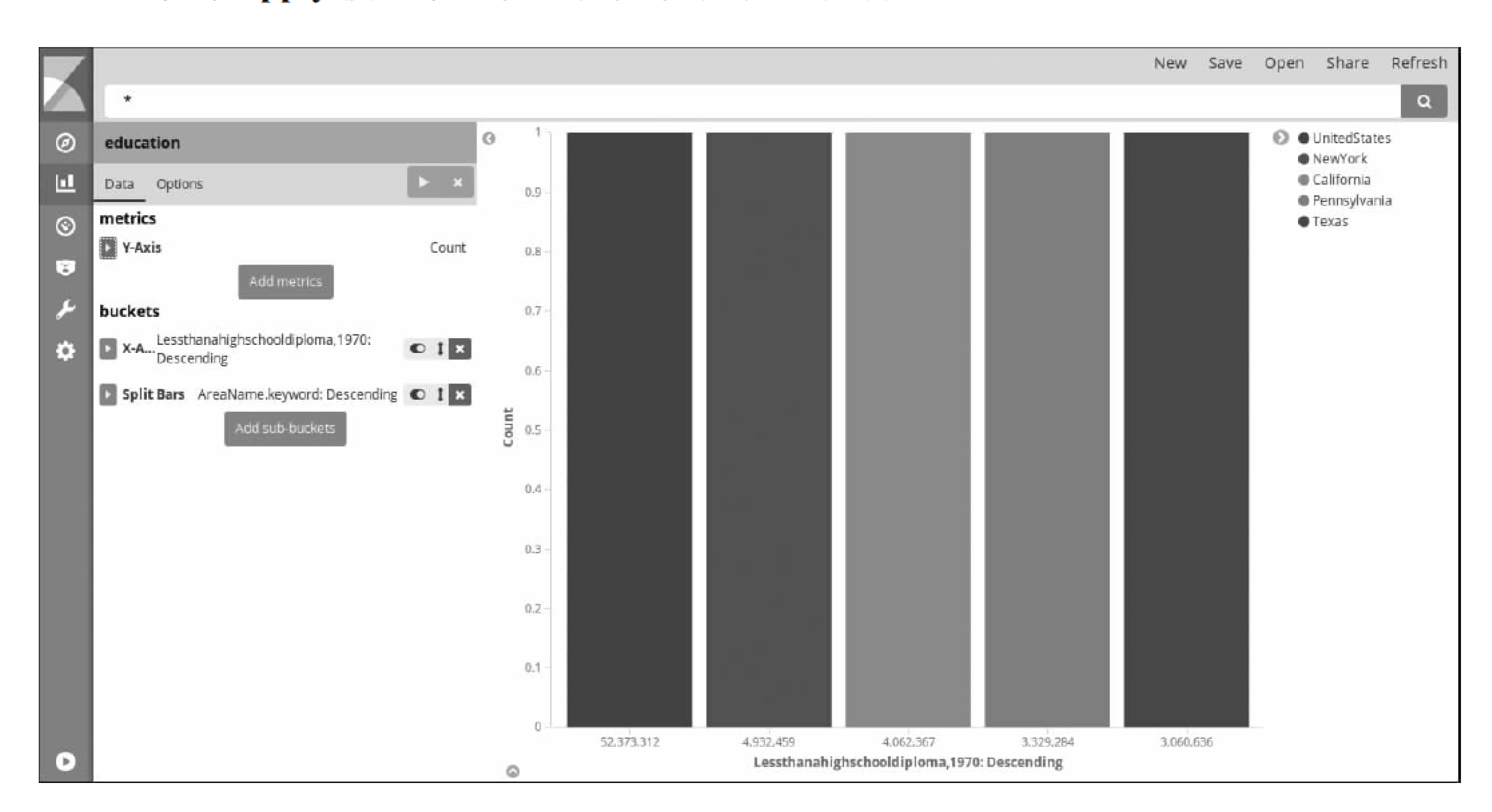
（1）单击 **Visualize** 选项卡并且选择垂直条形图来创建一个新的可视化内容。

（2）新建搜索并选择一个搜索源，指定索引名称为 education（在 Kibana 中配置多个索引的情况下）。

（3）在 **metrics** 部分，设置 **Y 轴**的聚合方式为 **Count**。在 **buckets** 聚合中，设置 **X 轴**的聚合方式为 Terms，并应用在 LessthanHighSchoolDiploma，1970 字段上。此处以词项出现的次数为排序依据，执行降序排序，并将 size 设置为 5。

（4）在子聚合中，设置 **Split Bars** 的聚合方式设置为 Terms，并应用在 AreaName.keyword 字段上。此处以 metric：Count 为依据降序排序，并将其 Size 设置为 5。

（5）单击 **Apply** 按钮来查看可视化结果，如下图所示。



将这个可视化统计图以 Top5AreaNamewithLessthanHSD1970 名称保存，这将用于创建面板。在这个可视化统计图当中，我们可以看到 AreaName 中 United States 的值，这是字段中所有值的总和，我们将它排除在其他可视化中。

4.12.3.4　1970 年学历达到中学毕业水平的前五个地区

在这个场景中，希望创建一个条形图，展示 HighSchoolDiploma1970 字段的前五个值及其所对应的地区名称。

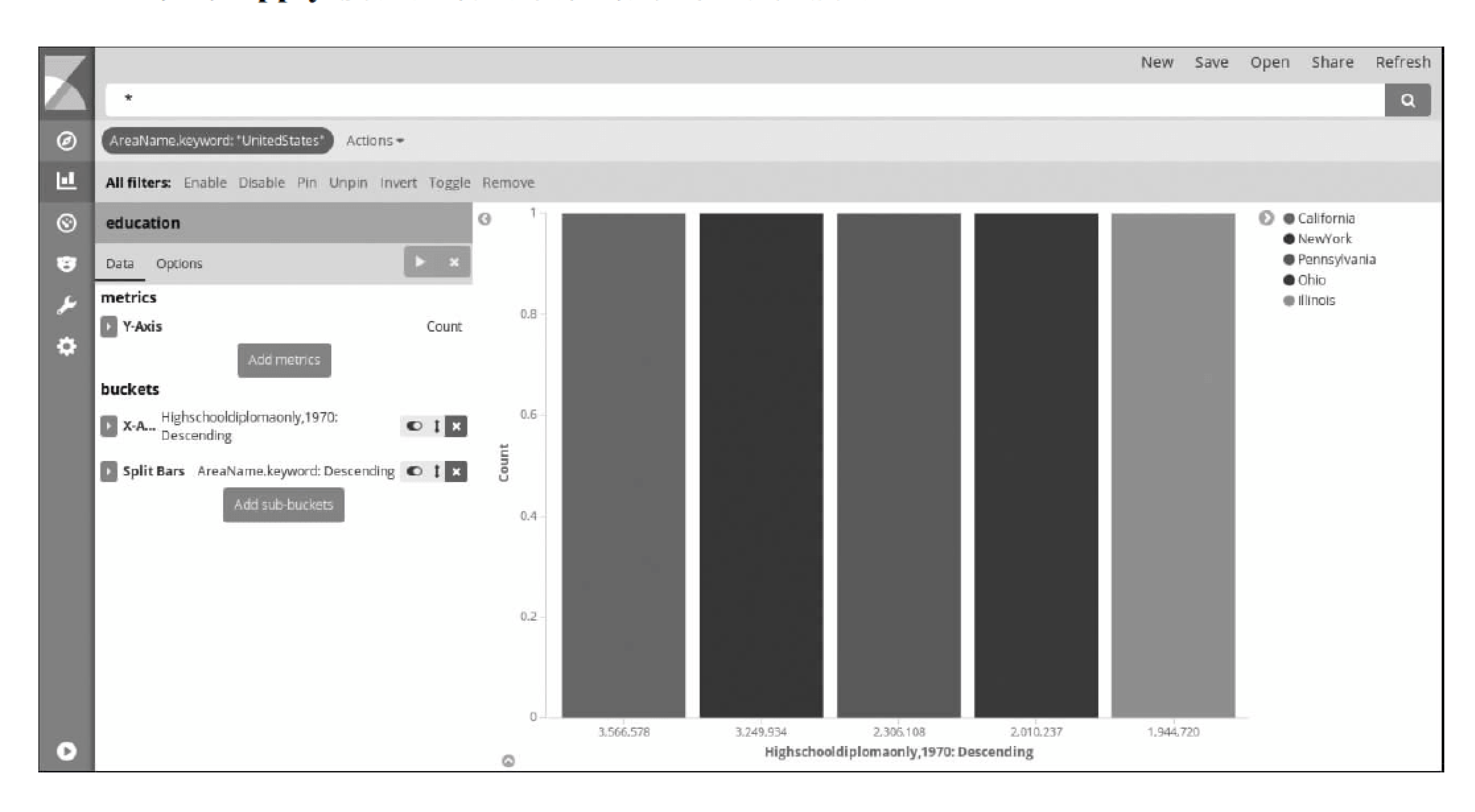
（1）单击 **Visualize** 选项卡，并且选择直方图创建一个新的可视化内容。

（2）新建搜索并选择一个搜索源，指定索引名称为 education（在 Kibana 中配置多个索引的情况下）。

（3）在 **metrics** 部分，设置 **Y 轴**的聚合方式为 **Count**。在 **buckets** 聚合中，设置 **X 轴**的聚合方式为 Terms，并应用在 HighSchooldiplomaonly，1970 字段上。此处以词项出现的次数为排序依据，执行降序排序，并将 size 设置为 5。

（4）在子聚合中，设置 **Split Bars** 的聚合方式为 Terms，并应用在 AreaName.keyword 字段上。此处以 metric：Count 为依据降序排序，并将其 size 设置为 5。现在需要排除 AreaNameunitedstates 的搜索结果，单击图例中的 **United States**，之后单击正过滤器符号来添加过滤器。然后选择过滤操作，并选择反转，它将显示除 United States 外的所有 AreaNames 信息。

（5）单击 **Apply** 按钮查看可视化结果，如下图所示。



将可视化文件保存为 Top5AreaNamewithHSD1970，该文件将创建一个新的面板。

4.12.3.5 1970 年按地区和州统计的低于高中毕业文凭的成人比例

在这个场景中，希望创建一个饼图，来展示 1970 年低于高中文凭的成年人中百分比最高的前五个值及其对应的地区名称：

（1）单击 **Visualize** 选项卡，并且选择饼图来创建一个新的可视化内容。

（2）新建搜索并选择一个搜索源，指定索引名称为 education（在 Kibana 中配置多个索引的情况下）。

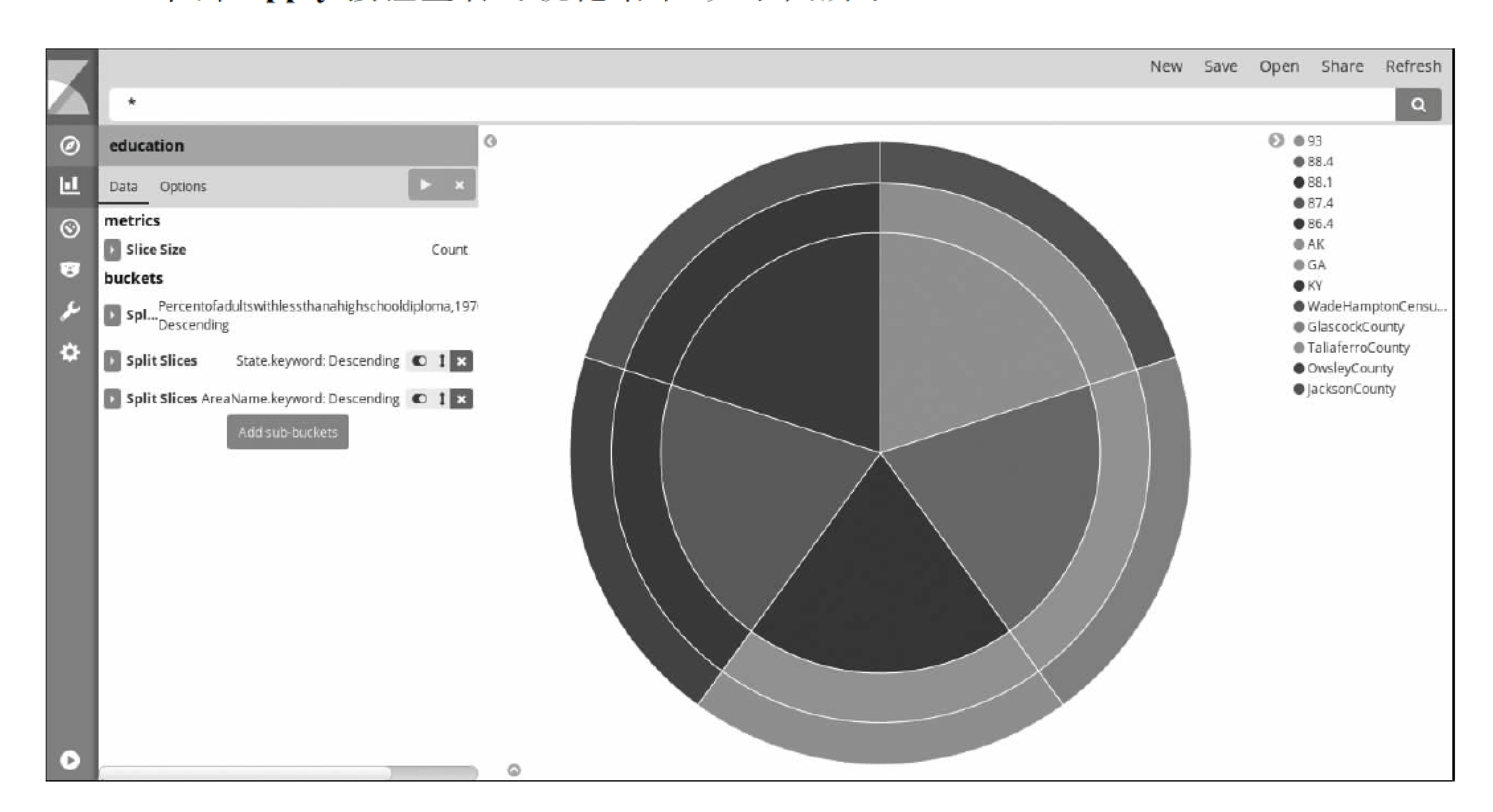
（3）在 **metrics** 部分，设置 **Slice Size** 的聚合方式为 **Count**。在 **buckets** 聚合中，选择 **Split Slices** 上的 Terms 聚合，并应用在 Percentofadultswithlessahighschooldiploma，1970 字段

上。此处以词项出现的次数为依据降序排序,并将其 size 设置为 5。

(4) 在子聚合中,设置 **Split Splices** 的聚合方式为 Terms,并应用在 State. keyword 字段上,以词项出现的次数为依据降序排序,并将其 size 设置为 5。

(5) 在子聚合中,设置 **Split Splices** 的聚合方式为 Terms,并应用在 AreaName. keyword 字段上。此处以词项出现的次数为依据降序排序,并将其 size 设置为 5。

(6) 单击 **Apply** 按钮查看可视化结果,如下图所示。



将可视化文件保存为%AdultslessthanHSD1970ByArea&State,该文件将用于创建面板。

4.12.3.6 根据计数以及 2013 年 RUCC 排名统计的前几个州

在这个场景中,希望创建一个数据表,来展示州字段的前五个值以及其对应的 2013 年 RUCC 的前五个值:

(1) 单击 **Visualize** 选项卡,并且选择**数据表**来创建一个新的可视化内容。

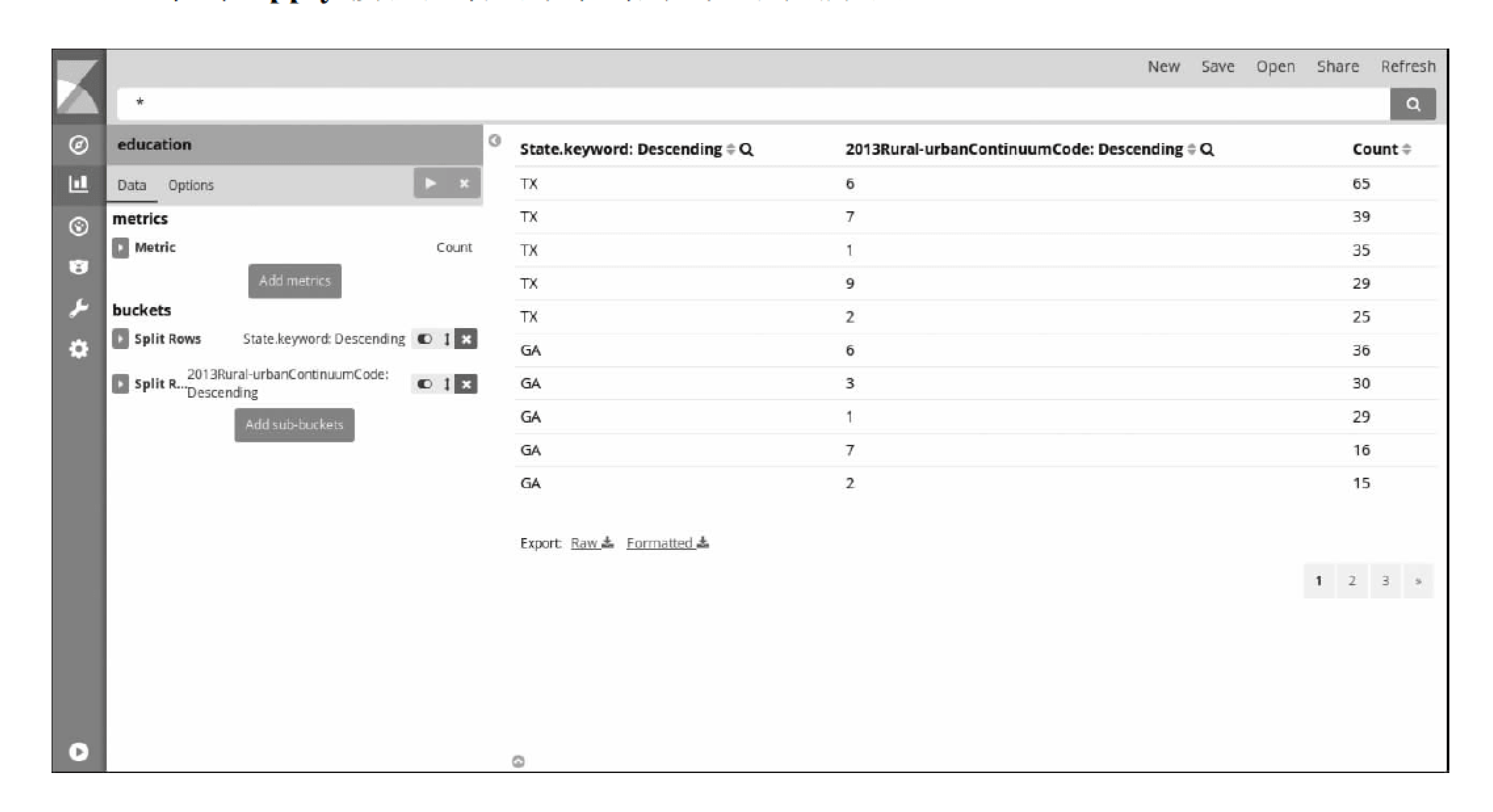
(2) 新建搜索并选择一个搜索源,指定索引名称为 education(在 Kibana 中配置多个索引的情况下)。

(3) 在 **metrics** 部分,设置 **Metric** 的聚合方式为 **Count**。在 **buckets** 聚合中,设置 **Split Rows** 的聚合方式为 Terms,并应用在 State. keyword 字段上。此处以 metric:Count 为依据来执行降序排序,并将 size 设置为 5。

(4) 在子聚合中,设置 **Split Rows** 的聚合方式为 Terms,并应用在 2013Rural-

urbanContinuumCode 字段上。此处以 metric:Count 为依据降序排序,并将其 size 设置为 5。

(5) 单击 **Apply** 按钮查看可视化结果,如下图所示。



将可视化文件保存为 Topstateswith2013RUCCCodes,该文件被用于创建面板。

4.12.3.7 了解统计数据

在这个场景中,希望创建一个一站式的有关数据统计的可视化内容。我们将看到记录的总数、2013 年 RUCC 的特定值、2013 年 UIC 的特定值、地区总数、地区名称总数、拥有低于高中文凭的成年人平均百分比、仅拥有高中文凭的成年人平均百分比、拥有大学或大专学历的成年人百分比、拥有学士学位或者更高学位的成年人的百分比。

(1) 单击 **Visualize** 选项卡,选择 **Metric** 创建一个新的可视化内容。

(2) 新建搜索并选择一个搜索源,指定索引名称为 education(在 Kibana 中配置多个索引的情况下)。

(3) 在 metrics 部分,设置 **Metric** 的聚合方式为 **Count**,并应用在 2013Rural urbanContinuumCode 字段上。

(4) 在 metrics 部分添加一个 Unique count 聚合,并应用在 2013UrbanInfluenceCode 字段上。

(5) 在 metrics 部分添加一个 Unique count 聚合,并应用在 State. keyword 字段上。

(6) 在 metrics 部分添加一个 Unique count 聚合,并应用在 AreaName. keyword 字段上。

(7) 在 metrics 部分添加一个 Unique count 聚合，并应用在 Percentofadultswithabachelor'sdegreeorhigher,2010-2014 字段上。

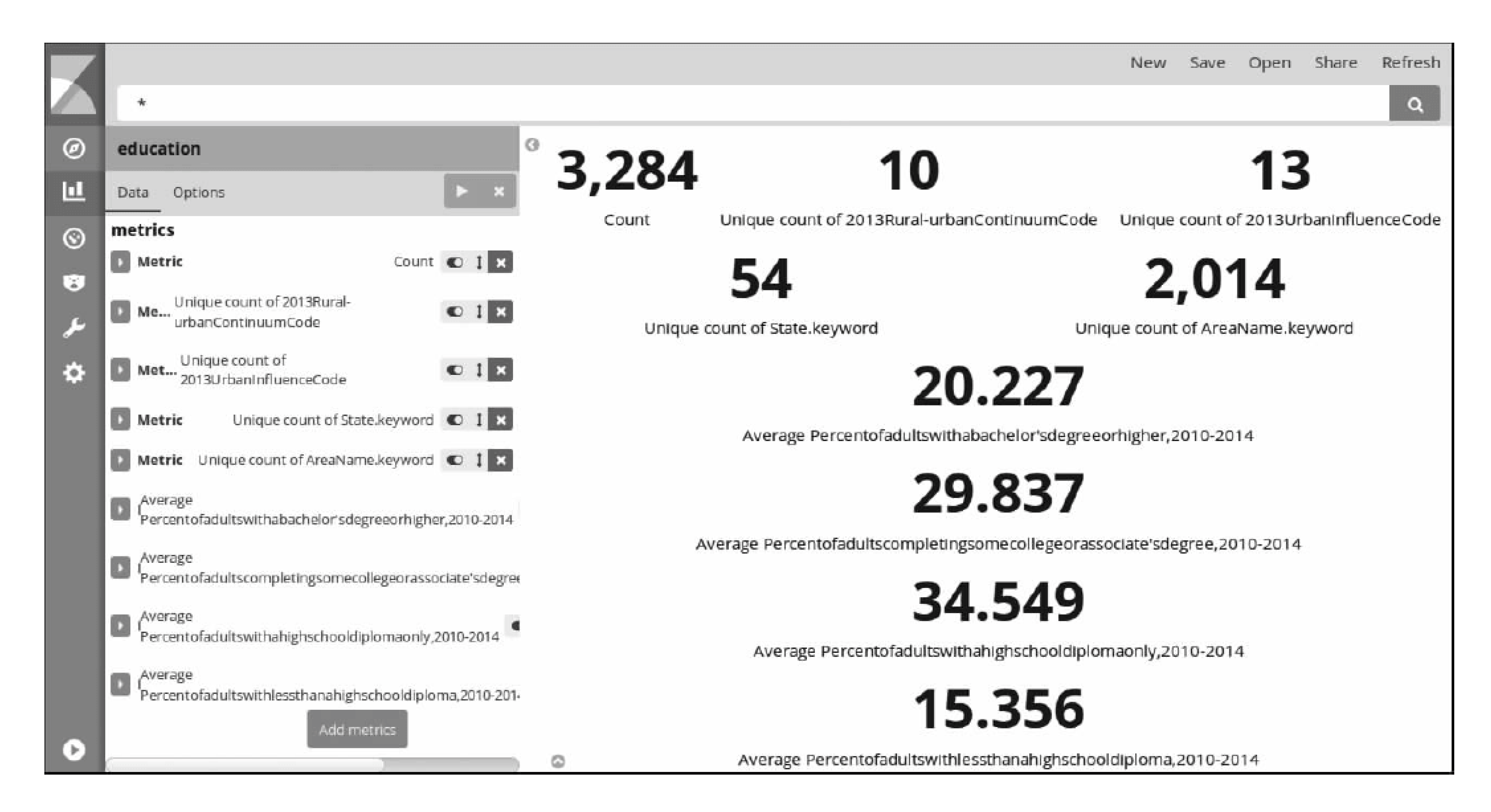
(8) 在 metrics 部分添加一个 Average 聚合，并应用在 Percentofadultscompletingsomecollegeorassociate'sdegree,2010-2014 字段上。

(9) 在 metrics 部分添加一个 Average 聚合，并应用在 Percentofadultswithabachelor'sdegreeorhigher,2010-2014 字段上。

(10) 在 metrics 部分添加一个 Average 聚合，并应用在 Percentofadultswithahighschooldiplomaonly,2010-2014 字段上。

(11) 在 metrics 部分添加一个 Average 聚合，并应用在 Percentofadultswithlessthanahighschooldiploma,2010-2014 字段上。在 **Options** 选项卡中，将**字号**(**Font Size**)改为 35，以显示所有信息。

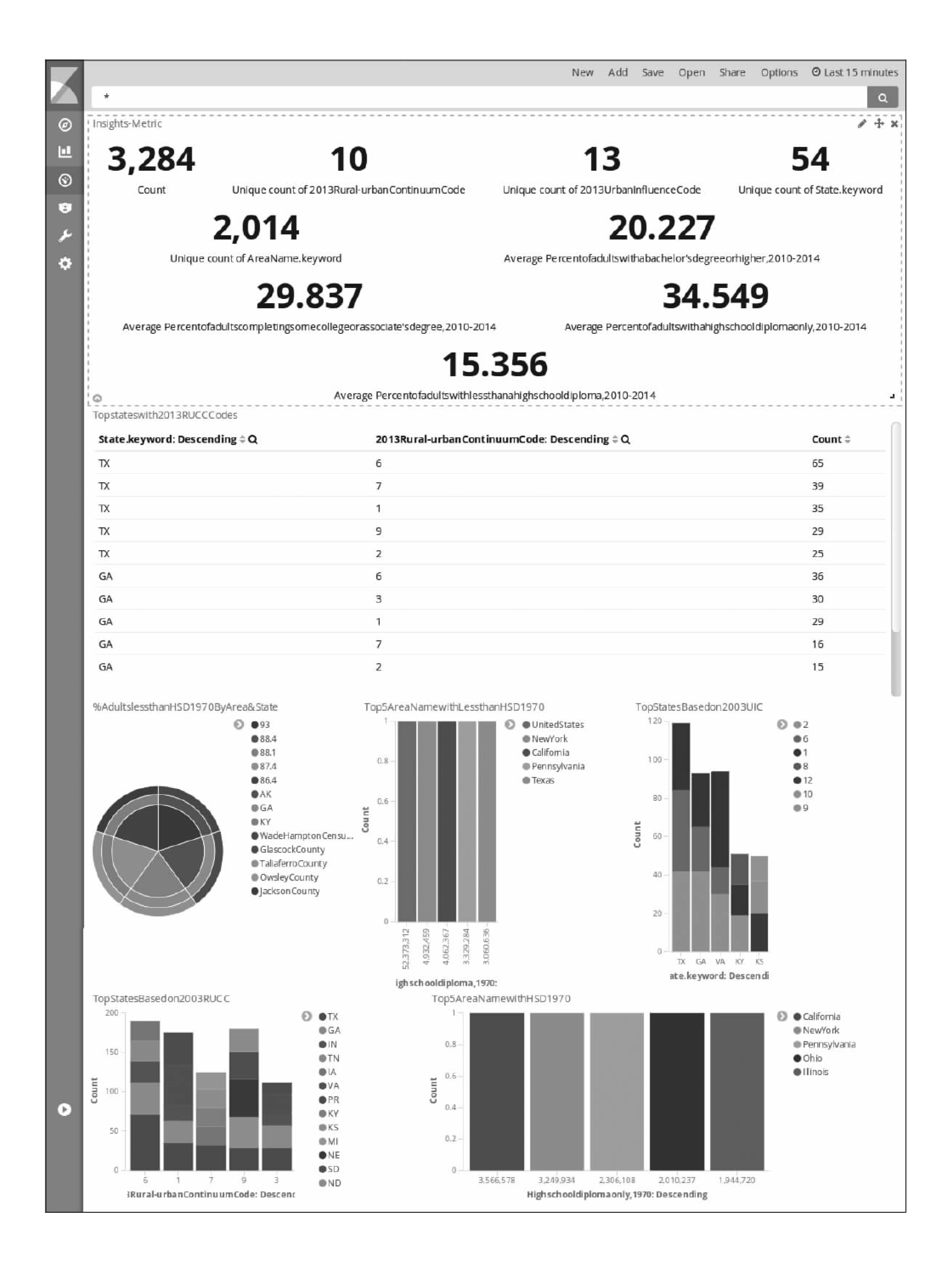
(12) 单击 **Apply** 按钮查看可视化结果，如下图所示。



将可视化文件保存为 Insights-Metric，该文件将被用于创建面板。

4.12.4 在 Kibana 中创建面板

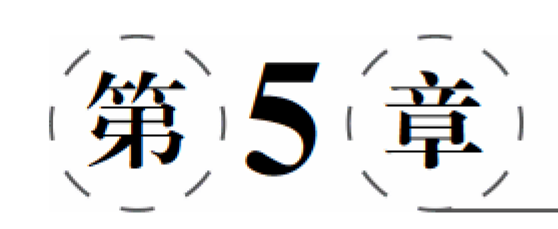
将所有已保存的可视化内容添加到**面板**。为了完成此操作，要进入 **Dashboard** 页面，并且单击**添加**(**Add**)。在添加了所有的可视化内容以及移动和调整大小之后，会得到一个如下图所示的面板。


New Add Save Open Share Options Last 15 minutes
Insights-Metric
3,284
Count
10
Unique count of 2013Rural-urbanContinuumCode
13
Unique count of 2013UrbanInfluenceCode
54
Unique count of State.keyword
2,014
Unique count of AreaName.keyword
20.227
Average Percentofadultswithabachelor'sdegreeorhigher,2010-2014
29.837
Average Percentofadultscompletingsomecollegeorassociate'sdegree,2010-2014
34.549
Average Percentofadultswithahighschooldiplomaonly,2010-2014
15.356
Average Percentofadultswithlessthanahighschooldiploma,2010-2014
Topstateswith2013RUCCCodes
State.keyword: Descending
2013Rural-urbanContinuumCode: Descending
Count
TX 6 65
TX 7 39
TX 1 35
TX 9 29
TX 2 25
GA 6 36
GA 3 30
GA 1 29
GA 7 16
GA 2 15
%AdultslessthanHSD1970ByArea&State
93
88.4
88.1
87.4
86.4
AK
GA
KY
WadeHamptonCensu...
GlascockCounty
TaliaferroCounty
OwsleyCounty
JacksonCounty
Top5AreaNamewithLessthanHSD1970
UnitedStates
NewYork
California
Pennsylvania
Texas
Count
52,373,312
4,932,459
4,062,367
3,329,284
3,060,636
TopStatesBasedon2003UIC
2
6
1
8
12
10
9
Count
TX GA VA KY KS
TopStatesBasedon2003RUCC
TX
GA
IN
TN
IA
VA
PR
KY
KS
MI
NE
SD
ND
Count
6 1 7 9 3
Top5AreaNamewithHSD1970
California
NewYork
Pennsylvania
Ohio
Illinois
Count
3,566,578
3,249,934
2,306,108
2,010,237
1,944,720
Highschooldiplomaonly,1970: Descending

4.13 本章小结

在这一章中，我们学习了 Kibana 及其界面，并且简单了解了 Kibana 的功能。通过一些例子，我们创建了许多类型的可视化统计图表来分析数据。如今，Kibana 在分析工具中非常受欢迎，因为它提供了众多功能。在下面的章节中，将学习更多的内容，例如实时数据集、生产环境服务器的日志分析等。

在下一章中，我们将会学习 Beats，它是 Elastic Stack 中的一个新组件。我们将以示例的形式来了解 Beats 的类型、使用方法以及其在 Elastic Stack 中的作用。



使用 Beats

在前面的章节中，探讨了以往的 ELK Stack 组件，读者已对每个组件的基础知识以及组件之间如何一起工作有所了解。现在，随着 Stack 的快速扩张与增长，来熟悉 Elastic Stack 中的最新组件。

在这一章中，将介绍 Beats 的基本概念以及 Beats 的其他内容，从基础到进阶层面来展示 Beats 的各种类型及其功能，并讲解 Beats 是如何融入 Elastic Stack 中的。

本章主要内容如下：

- Beats 组件介绍；
- Beats 与 Logstash 的不同之处；
- Beats 如何融入 Elastic Stack；
- 不同类型的 Beats 组件概述；
- Elastic 团队开发的 Beats 组件；
- 社区成员开发的 Beats 组件；
- Elastic Stack 上的 Beats 实战。

5.1 Beats 简介

在了解 Beats 之前，先了解 Beats 是如何出现的。Beats 不是作为 ELK Stack 背后的 Elastic 公司的一个项目开始的，而是始于 Packetbeat 项目。在这里，开发人员访问并统计各服务器之间的通信情况，并通过数据传输为其提供信息。

他们想开发一个解决方案，将数据存储在 Elasticsearch 中，但不会消耗很高的内存或资源。之前他们在每台需要获取信息的服务器上运行 Logstash 程序时，遇到了 JVM 消耗内存和资源过大的问题。而 Packetbeat 因其轻量级的数据传输能力广受欢迎，它可以将数据传输到 Elasticsearch 而无须消耗太多的内存和 CPU 资源。随着受欢迎程度的上升，Elastic 收购了 Packetbeat 并创建了 Beats 平台。这一平台包含许多由 Elastic 开发的 Beats 组件；这其中也包含由社区开发者开发的 Beats 组件，它们的软件框架和类库均已开放。

Beats是最新添加到Elastic Stack的组件。Beats组件不像ELK Stack中的其他组件那样具备即插即用的功能,例如ELK Stack使用Elasticsearch来搜索和分析数据,使用Logstash来处理日志,使用可视化工具Kibana来将数据进行可视化展示。然而,Beats整体是一个提供各种不同的Beats组件的平台。它拥有解析和传输数据的能力,能够从不同来源收集日志,经过解析后传送至Logstash或Elasticsearch,然后在Kibana中进行可视化。因此,Beats是一种开源的轻量级数据传送工具,可以收集主机或服务器中的操作数据并发送到Elasticsearch或Logstash中。可以将数据从Beats直接发送到Elasticsearch,或者出于不同的目的,先将数据发送给Logstash进行处理和分析,之后再传入Elasticsearch。也可以将Beats理解为收集数据并传送到Elasticsearch或Logstash的轻量级守护进程的集合。由于Beats组件轻量的本质,因此它不会在操作系统中占用很大空间。

Beats平台中包含了许多针对不同目的而开发的Beats组件。如果要解决具体的用例,就可为用例创建一个beat程序,它将带来灵活性,之后也可以公开这个组件为开源软件社区做出贡献。为了简化自己的Beats组件的开发过程,Beats平台提供了API中的所有选项和使用方法,这样就可以亲自开发beat程序,其间只需关注特定于具体应用的beat的处理逻辑,而其他一切都交由平台处理。平台提供了一个处理各种服务的通用框架,并为每个人提供了统一的软件框架来使用和开发Beats。Beats平台使用Go语言编写,Go语言通过调用更少的资源和内存使Beats组件更加高效,而这样反过来又有利于它基于代理agent的数据传输。

5.2 Beats与Logstash的不同之处

从我们所了解的Beats来看,它与Logstash十分相似。其中我们使用Input插件来解析数据。那么,它们哪里不一样呢?Logstash只是数据分析工具,而Beats是一套数据传输工具,可以将数据从文件、数据流或日志等一系列输入源中传输数据。尽管Logstash也能传输数据,但这不是它的主要用途。简而言之,Beats和Logstash在功能上很相似,但两者在开发方式和使用的底层技术上又有着明显的差异。

让我们看看它们之间的区别:

- Logstash会消耗大量的内存,并且需要较多的系统资源;Beats需要较少的系统资源,消耗的内存也很少。
- 安装在需要收集日志的操作系统上的Logstash是一套体量相对较大的软件;Beats则是轻量级的数据传输工具,它可以在多个操作系统之间传输数据。
- 如果从多个系统收集日志,则不需要在每个操作系统中运行Logstash服务;Beats可以在所有需要传输数据的操作系统之间运行。

- Logstash 是基于 Java 语言编写的，它需要 JVM 的支持，而 Beats 是基于 Go 语言开发的。
- 有时候只需要解析一些日志，而不使用 Filter 插件来做任何形式的处理。在这样的情况下，可编写一段 Logstash 配置。为了编写这些配置，需要事先学习配置文件的编写方法，而这就需要事先学会在 Elasticsearch 中存储数据。然而，只计划收集日志的时候，还是推荐使用 Beats 组件。这样既避免了额外的数据处理过程，又能将数据直接传输到 Elasticsearch 中，不会包含增加程序复杂度的其他工作插件。

根据前面的解释，我们可以了解到，Logstash 是一个体量比较重的工具，将它安装在多个操作系统中，需要占用大量的内存空间，同时会消耗较多的系统资源。为解决这个问题，我们有 Logstash-forwarder，它是针对与 Beats 相同的应用场景建立的，是一套轻量级的工具，不需要很多的系统资源。那么 Logstash-forwarder 和 Beats 一样吗？

Logstash-forwarder 的确是针对与 Beats 相同的用例建立的，它是将数据从多个数据源传输到 Logstash 或 Elasticsearch 的唯一一个轻量级解决方案。此外，它曾在社区中被忽视，在添加新功能、改进现有功能和修复 bug 方面比较落后。现在，随着时间的推移，Elastic Stack 已被越来越多的组织和企业使用，人们对定制个性化、轻量级数据传输工具，并实现向 Elasticsearch 或 Logstash 传输数据功能的需求，在日益增长。

作为 Beats 组件的一员，Filebeat 是基于 Logstash-forwarder 的项目，并利用其现有项目代码开发。现有的项目代码已经使用 Go 语言重新封包，并遵循公共软件框架将其开发为一款 beat 程序，同时使用同样的公共框架来开发其他 beat 程序。Logstash-forwarder 的核心功能已在 Filebeat 中使用，比如发送消息后接收确认信息以防止数据的丢失，以及 Lumberjack 协议等，已被广泛应用在 Filebeat 中。Beats 框架服务使 Beats 更容易作为 Windows 服务来运行。

这样可能会出现一个问题：Beats 会不会在未来取代 Logstash 呢？这个问题的答案很简单：不会的。Logstash 类似于功能多样的“瑞士军刀”，能提供从多个数据源加载数据的功能，使用各种强大的插件来处理日志，并提供将多个来源的输出数据进行存储的功能。简而言之，Logstash 提供数据 ETL(数据的提取、变换和加载)的功能；而 Beats 是轻量级的数据传输工具，能将数据传输到 Logstash 或 Elasticsearch 中，其间没有对数据进行任何转换。

5.3 Beats 如何融入 Elastic Stack

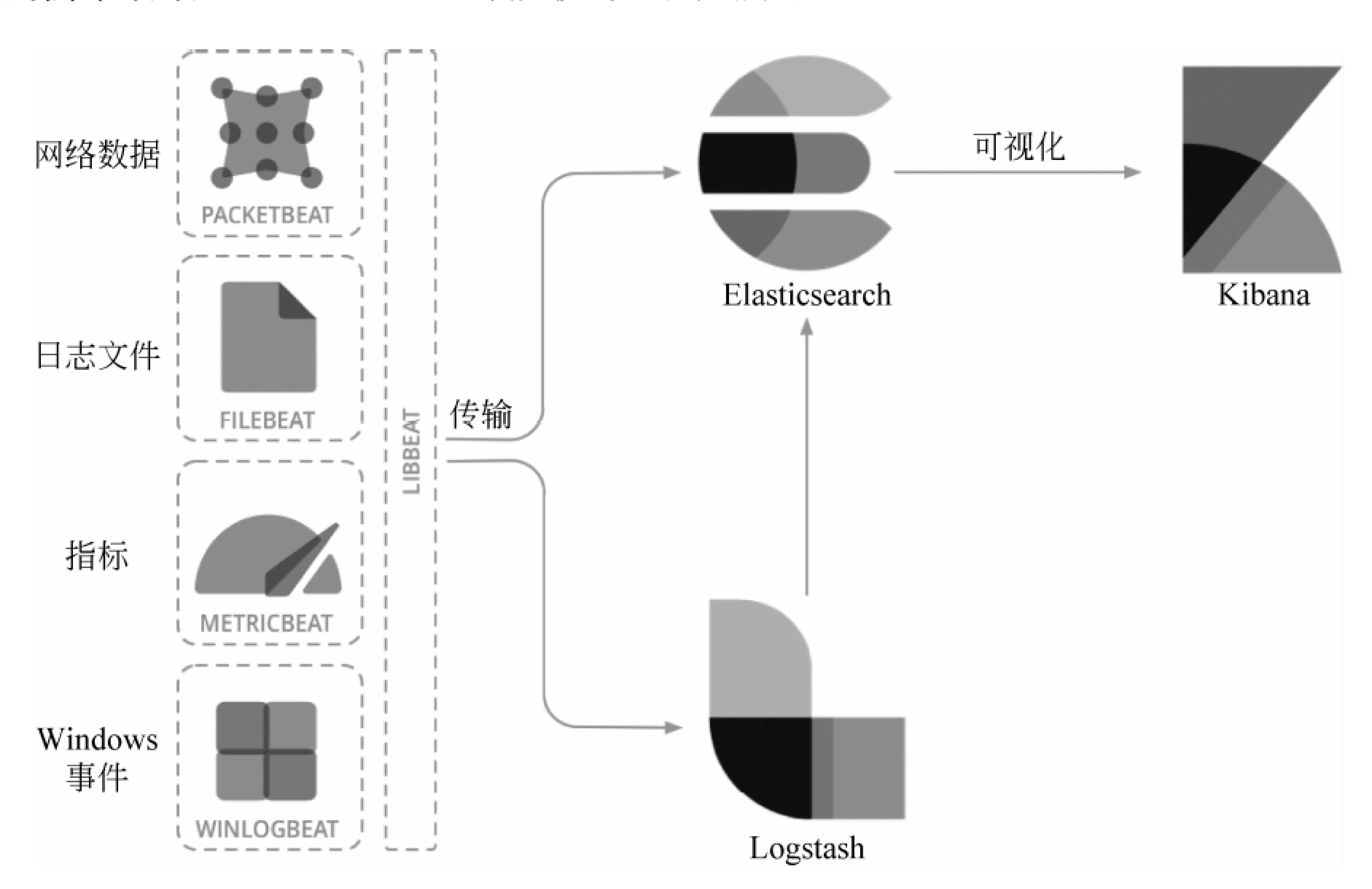
使用 ELK Stack 一段时间之后，我们会问：Stack 中还可以加入其他组件吗？

我们认为这是最大问题之一：在哪里以及如何将 Beats 融入 Elastic Stack？在探讨了

什么是 Beats 组件之后，本节将涉及针对这个问题的答案。

Beats 组件会为使用过程带来无穷无尽的机遇，它已经作为核心组件之一被添加到 Stack 当中。在 ELK Stack 中，会受到 Input 插件的限制，只有通过这些插件才能读取数据。如果想在 Elasticsearch 中为操作数据创建索引，例如多操作系统之间的事务信息、Docker 容器统计信息、Tomcat JMX 数据或系统进程级别的统计数据，就需要写一个 Logstash 的 Input 插件，然后将其用于这样的场景。在一些使用场景中，要从多个操作系统中获取日志，这时可以使用 Beats 来收集这些数据，并传输到一个集中的 Logstash 服务器来解析、处理和存储数据。这种场景不需要使用具有大容量内存和足够资源的系统，因为 Beats 组件是轻量级的数据传输工具。此外，在只需要分析数据而不用处理数据的使用场景中，Beats 组件将会发挥作用，可以将数据直接存储在 Elasticsearch 中，而不需要用到 Logstash。

让我们来看看 Elastic Stack 的概貌，如下图所示。



参考链接：https://www.elastic.co/guide/en/beats/libbeat/5.1/images/beats-platform.png。

在前面的图表中，可以看到 Beats 平台由 Elastic 开发的不同 Beats 组件构成。Beats 组件负责收集和传输数据到 Logstash 或 Elasticsearch 中，以进行进一步处理或存储。之后，这些数据可以用来分析运营数据或使用 Kibana 创建可视化统计图表或面板。

理解 Beats 之后，让我们来看看由 Elastic 和由社区开发者开发的各种 Beats 组件。

5.4 不同类型的 Beats 组件概述

在本节中，将讨论已经开发并在企业中用于开发和生产目的的 Beats 组件。Beats 组件是由 Elastic 以及社区开发者开发和优化、提升性能的。

下面是由 Elastic 开发的 Beats 组件：

- Packetbeat；
- Metricbeat；
- Filebeat；
- Winlogbeat；
- Libbeat。

下面是开源社区开发的 Beats 组件：

- Apachebeat；
- Dockbeat；
- Elasticbeat；
- Execbeat；
- Httpbeat；
- Lmsensorsbeat；
- MySQLbeat；
- Twitterbeat 以及更多的 Beats 组件。

> Elastic 不会为开源社区开发的 Beats 组件提供任何支持。

5.4.1 Elastic 团队开发的 Beats 组件

如前所述，我们将研究 Elastic 团队提供的不同类型的 Beats 组件，了解每种 beat 程序的含义以及它们提供的功能。

5.4.1.1 Packetbeat

Packetbeat 是主要的 Beats 组件，它起始于 Elastic 公司外的一个项目。Elastic 为创建 Beats 平台，将开发该项目的公司收购，现在它已经成为 Elastic Stack 不可或缺的一部分。

Packetbeat，正如从其名字可猜到的那样，是一个提供网络中不同服务器程序间相关事务信息的数据包分析器。它提供了不同类型的应用服务器之间网络通信相关的统计信息。

5.4.1.2 Metricbeat

Metricbeat 是重要的 Beats 组件，它收集多个操作系统间以及系统服务程序的运行指标(metrics)。在 Linux 系统中它是 top 命令的扩展[①]。top 命令是用来在系统层级上获取进程信息的命令，而 Metricbeat 则是该命令的一个扩展，可以查看进程所消耗的资源(如内存使用量、CPU 利用率)，以及其他系统信息。

除了在 Linux 系统中用 top 命令获取系统运行指标之外，这一款 Beats 组件还可以收集其他指标信息，并将这些信息一并传输到 Elasticsearch 或 Logstash 中去，之后使用 Kibana 创建出精美的可视化统计图表。Metricbeat 是一款轻量级的数据传输工具，它每隔一段时间就会获取和传输操作系统和相关服务程序中的运行指标。

5.4.1.3 Filebeat

Filebeat 组件的灵感来自于 Logstash-forwarder 项目，它已经被企业在生产环境中用了很长时间。通过以全新的方式对 Logstash-forwarder 重新封包，为 Filebeat 的发展铺垫了道路。

Filebeat 是一个开源的日志传送工具，传输来自多个操作系统的日志，且消耗最少的资源。它将日志文件从操作系统传输到 Elasticsearch 或 Logstash 中，提供了将日志从多个操作系统传输到集中的操作系统或服务器的功能，从中可以对日志进行解析和处理。它没有任何依赖项或任何插件来管理。

5.4.1.4 Winlogbeat

Winlogbeat 是 Elastic Beats 平台中最新添加的组件。这种 beat 组件之所以出现，是出于专门获取 Windows 事件日志的需求。Winlogbeat 有助于从不同类型的 Windows 事件中获取各种事件日志，这是使用这款 Beats 组件的一个主要目的，它十分简单易用。

Winlogbeat 是一个开源的数据传输工具，可以在多个操作系统中传输 Windows 事件日志数据。日志文件可以按要求传输到 Elasticsearch 或 Logstash 中。它可以作为 Windows XP 或更高版本操作系统的服务程序安装到系统中。

5.4.1.5 Libbeat

Libbeat 不像之前讨论过的 Beats 组件，它不提供任何可用的功能。这是一个由 Elastic 团队创建的库，是一个可以对其他 Beats 进行开发的通用框架或架构。这个库是用 Go 语言编写的，它公开了可以被所有 Beats 组件直接使用的 API，从而减少了开发工作，并确保所

---

① 译者注：top 命令是 Linux 下的性能分析工具，能实时显示系统中各进程的资源占用状况。

有的 Beats 都以类似的方式运行，这样就很容易使用常用工具打包和运行。

开发 Beats 组件常用的部分都在 Libbeat 库中提供，其中包含提供各种功能的软件包，如向 Elasticsearch 或 Logstash 发送数据，实现记录日志，配置输入，处理通知，处理 Windows 服务等。对 Libbeat 库的优化将有利于所有的 Beats 组件，因为开发者可以使用官方提供的新库而不必重写代码。

5.4.2 社区开发者开发的 Beats 组件

在这一节中，将看到一些由社区开发者开发并对 Beats 平台做出贡献的 Beats 组件。我们将看到其中的两个组件，(通过它们)了解 Beats 是用来做什么的，以及它们提供了哪些功能。

5.4.2.1 Dockbeat

由于容器类应用程序 Docker 的不断普及，开发者在开源社区贡献了 Dockbeat 组件。随着 CI 工具的普遍使用，它变得越来越流行。在生产中 Docker 正被越来越多地使用，拥有这样一种提供容器相关统计信息的监测工具是十分必要的。如果使用为 Beats 平台提供通用框架的 Elastic Stack，那为什么还要创建一个单独的工具，通过 Elastic 的团队在 Libbeat 库中提供的功能来处理数据呢?

Dockbeat 提供容器的统计信息，如容器的属性、CPU 使用率、网络运行状况统计信息、内存消耗量统计信息、I/O 操作的访问统计、Dockbeat 状态信息等。如同其他的 Beats 组件一样，这是一个安装在多个操作系统中的轻量级工具，它可以周期性地读取 Docker 容器的统计信息，并将这些信息保存在 Elasticsearch 的索引中。我们可以使用 Kibana 将这些存储的信息进行可视化。

5.4.2.2 Lmsensorbeat

Lmsensorbeat 是一款性能惊人却又简单易用的 Beats 组件，利用系统核心模块调用 lm-sensors 库，通过来自传感器的信息提供系统相关的信息(提供了检测系统中多种传感器信息的命令，这些命令可以通过安装 lm-sensors 库来执行)。

Lmsensorbeat 通过监控各种类型的传感器来收集信息，如 CPU 或主板的温度、电压、散热风扇转速等。这是经常会被人忽略的基本信息，但现在很容易监控这些信息，并在 Elasticsearch 中为这些信息创建索引。

5.5 Elastic 团队开发的 Beats 组件

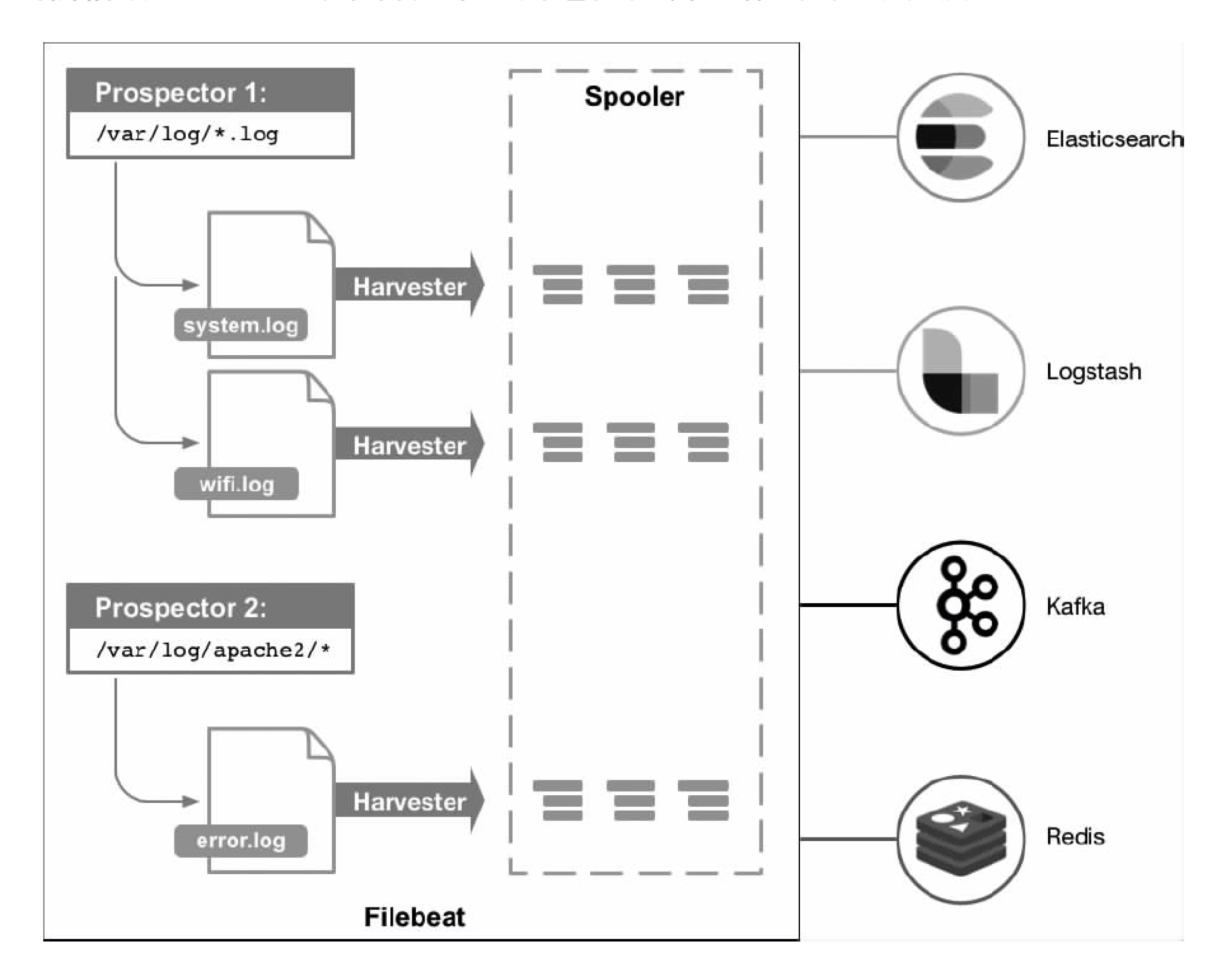
我们已经学习了 Elastic 团队提供的不同类型的 Beats 组件。在这一节中，将学习这些 Beats 组件并了解它们的特性，学习如何设置它们，以及如何根据各种配置选项来配置

Beats 组件。本节将提供详细了解 Beats 组件所需的必要信息。

5.5.1 了解 Filebeat

如前面所讨论的那样，Filebeat 是一个开源的日志传输工具，它的灵感来自于 Logstash-forwarder 项目，它也是基于 Logstash-forwarder 的源代码开发的。Filebeat 用于监控日志信息（文件或目录）并将那些日志传送到 Logstash 中进一步处理，或传输到 Elasticsearch 中为这些数据创建索引。

让我们借助 Filebeat 的架构图来了解它是如何工作的，如下图所示。



链接：https://www.elastic.co/guide/en/beats/filebeat/5.1/filebeat-overview.html。

在上图中，需要理解很多新的术语，如 **Prospector**、**Harvester** 和 **Spooler**。下面让我们试着解释这些术语及其含义：

- **Prospector**：这里包含了所有要被读取的日志存储路径列表。这些路径是使用配置文件配置的。
- **Harvester**：当日志信息被 Prospector 获取时，这一功能将被 Filebeat 启动。Harvester 的作用是读取设置并检测日志文件中出现的新内容，然后收集这些内容并发送到 Spooler。
- **Spooler**：负责从 Harvester 接收输入信息。Spooler 处于核心位置，其中所有日志信

息都在这里累积和聚合。经过聚合的数据将会依照配置成为 Filebeat 组件的输出。

每当 Filebeat 启动时，它创建一个或多个 Prospector，然后开始寻找路径中指定的日志文件。它在指定的路径中查找所有的位置来读取数据。每一个文件被 Prospector 找到之后，相应的 Harvester 将会被启动。每个 Harvester 只读取单个文件的内容。如果找到多个文件，则启动多个 Harvester。之后每一个 Harvester 对相关联的文件进行读取（逐行读取），并发送文件内容到 Spooler。接着，Harvester 在相关联的日志文件中等待新的内容出现（如果有的话）。一旦发现新的内容，它就会把新的日志文件数据送到 Spooler 中。Spooler 负责将多个 Harvester 中的数据进行聚合，经过聚合的数据将会依照配置被处理为 Filebeat 的输出。

> Harvester 负责文件的打开和关闭。同样，如果一个文件在被 Harvester 读取的同时被删除或重命名，Filebeat 就会将其所在磁盘空间保留到 Harvester 读取后关闭为止。

了解 Filebeat 如何工作之后，我们来安装和使用 Filebeat 组件。

可以参阅第 1 章 Elastic Stack 概述中的安装 Filebeat 部分来安装 Filebeat。

让我们来看看 Filebeat 的配置文件来了解和使用 Filebeat 组件不同部分的配置。

Filebeat.yml 文件的内容分为以下几个部分：

- Filebeat Prospectors；
- Filebeat Global Options；
- Processors Configuration；
- Output Configuration；
- Paths Configuration；
- Logging Configuration。

Filebeat Prospectors 这一部分列出了依照配置信息来获取数据的 Prospectors 列表。Filebeat Global Options 包含 Filebeats 中常用的属性。Processor Configuration 中包含所有与处理器相关的属性。Output Configuration 包含将数据从 Beats 组件输出到多种可用组件的配置信息，输出的组件可以选择 Elasticsearch、Logstash、Kafka、Redis、File 和 Console 等。Paths 部分提供默认的 Filebeat 安装路径的配置信息。Logging Configuration 部分提供了多种设置项来获取日志级信息，这些信息可以用来调试程序。

我们来探讨上面几节中提到的配置属性。

5.5.1.1 Filebeat Prospectors 配置

以下示例配置文件包含许多属性。让我们看看其中的一部分：

```
filebeat.prospectors:
- input_type: log
  paths:
  - /var/log/*.log
  #encoding: plain
  #exclude_lines: ["^DBG"]
  #include_lines: ["^ERR", "^WARN"]
  #document_type: log
  #scan_frequency: 10s
  #max_bytes: 10485760
```

> ℹ 默认情况下，在 filebeat.full.yml 配置文件中，所有以 # 开头的属性都会包含固定的预设值。如果想修改属性值，可去掉属性名称前面的 # 号并设置属性值。

下面解释每个属性的含义，这样可以很容易熟悉这些属性：

- input_type：指定输入数据以何种方式被读取。它被用来指定将事件发布到 Elasticsearch 或 Logstash 的输入类型。这项属性包含 log（读取日志文件的每一行）和 stdin（读取标准输入）两种值。它的默认值是 log。
- paths：Filebeat 最初需要一个路径，并在该路径下搜索日志文件，其中每一条路径都作为一个搜索日志文件的 Prospector。对于每一个被找到的日志文件，Prospector 都会创建一个 Harvester 从文件中读取数据。在 paths 属性中设置的路径定义了 Filebeat 搜索.log 日志文件的目录。如果要指定多个目录，则只需在新一行中添加一个“-”号，后面跟另一个目录的名称即可。每一行都可以指定一个单独的路径。
- encoding：Filebeat 使用的文件编码。它用来读取包含国际字符的文件。这里使用的编码名称是 **W3C（万维网联盟）**推荐的。可以指定的编码有 UTF-8、BIG5、latin1、plain 等。
- exclude_lines：指定文件中 Filebeat 要忽略的行，可以使用正则表达式来指定。文件的内容也可以指定多行，多行信息在被 exclude_lines 过滤之前会被合并为单行。在配置文件中，如果正则表达式的值设定为“^DBG”，那么 Filebeat 就会忽略以字符串"DBG"开头的所有行。
- include_lines：指定文件中 Filebeat 要处理的行，可以使用正则表达式来指定。文件的内容也可以指定多行，多行信息在被 exclude_lines 过滤之前会被合并为单行。在配置文件中，如果正则表达式的值设定为["^ERR", "^WARN"]，那么 Filebeat 就会只处理以字符串"ERR"或"WARN"开头的行。

> ℹ在 Filebeat 中总是先执行 include_lines，之后再执行 exclude_lines。

- document_type：如果将数据输出到 Elasticsearch，那么这项属性用来指定索引的类型。这里指定的属性值将会成为索引中类型的名称。它的默认值是日志 log。
- scan_frequency：指定 Prospector 监测新文件出现的时间间隔，以及 Harvester 监测文件中新内容出现的时间间隔。设置这项属性之后，指定的目录和子目录中任何新创建的文件都会被扫描。它的默认值是 10s。
- max_bytes：指定 Filebeat 能够接收的单个事件日志的最大字节数。它的默认值被指定为 10485760，也就是 10 兆字节。如果任何一个事件日志超过了指定的字节数，那么多出的字节数将会被丢弃，从而不会被 Filebeat 获取。

5.5.1.2 processors 配置

Libbeat 库为 processors 提供了一个通用平台，可以在多种 Beats 组件中无缝使用。processors 并不是什么新东西，它是附加了新功能的过滤器。

processors 可用于：

- 减少输出数据中导出字段的数量；
- 使用元数据和附加处理功能优化事件。

每个 processor 都被赋予一个事件作为输入，接着执行 processor 的条件(condition)或操作(action)，最后将处理过的事件作为输出。多个 processor 也可以被这样定义和使用。

> ℹ Filters 已改名为 processors，这里的 processors 在功能上与 Elasticsearch 中的 Ingest Node processors 是不同的。

要定义一个 processors，需要指定 processor 的名称、它的条件(该项为可选项)，以及要在 processors 中应用的参数，语法是：

```
processors:
 -<processor_name>:
   when:
       <condition>
   <parameters>
```

> ℹ可以指定一个或多个 processors，程序将按照它们在配置文件中定义的顺序执行。

processors 支持的条件有：

- equals：检查值是否与指定的字段值匹配。它只接受整数或字符串值。
- contains：检查该值是否为前面所指定字段的一部分。它只接受字符串字段或字符串字段数组中的字符串值。
- regexp：检查正则表达式是否为指定字段的一部分。它只接受字符串值。
- range：检查字段是否在指定值范围内。它支持 LT(小于)、GT(大于)、LTE(小于等于)和 GTE(大于等于)的条件。它只接受整数或浮点值。

这里还有一些操作符使用了之前章节中提到的条件，即 OR、AND 和 NOT 操作符。它们用于一次指定多个条件的情况。

支持的 processors 有：

- add_cloud_metadata：从支持的云端服务提供商(EC2、Digital Ocean 和 GCE)添加元数据信息。语法如下：

```
processors:
  -add_cloud_metadata:
```

- decode_json_fields:解码 JSON 字符串。语法如下：

```
processors:
  -decode_json_fields:
    fields: ["field1", "field2", ...]
    process_array: false
    max_depth: 1
```

在这里，fields 是必选设置，其余是可选的。

- drop_event：删除符合条件的事件。它在执行时需要一个条件，否则所有事件都将被删除。语法如下：

```
processors:
  -drop_event:
    when:
    condition
```

- drop_fields：删除事件中符合条件的字段。它在执行时如果缺少条件，那么所有字段都将被删除。语法如下：

```
processors:
  -drop_fields:
    when:
    condition
  fields: ["field1", "field2", ...]
```

> ℹ即使在字段 fields 下指定，时间戳@timestamp 和类型的字段也不能被删除。

- include_fields：在条件匹配时容纳事件中的字段。如果缺少条件，则只导出指定的字段。可以在 processors 部分指定一个或多个 include_fields。语法如下：

```
processors:
  -include_fields:
    when:
      condition
    fields: ["field1", "field2", ...]
```

> ℹ即使在字段中没有指定，也会包含时间戳@timestamp 和类型 type 字段。

5.5.1.3 output 配置

output 配置部分包含将输出数据发送到输出组件的选项，可以指定一个或多个输出组件。正如前面提到的，可以发送输出的数据到 Elasticsearch、Logstash、Kafka、Redis、File 和 Console 中。

1. 配置 Elasticsearch 的 output

这里是一些 Elasticsearch 输出选项：

```
output.elasticsearch:
    #enabled: true
    hosts: ["localhost:9200"]
    #index: "filebeat-%{+yyyy.MM.dd}"
    #proxy_url: http://proxy:3128
    #max_retries: 3
```

让我们尝试解释每个属性的含义，这样可以很容易熟悉这些属性：

- enabled：启用或禁用指定的输出模块。
- hosts：指定要连接的 Elasticsearch 节点地址，可以根据需要连接到一个或多个 Elasticsearch 节点。日志事件在多个 Elasticsearch 节点中是以轮转的顺序分布的。每个 Elasticsearch 节点可以定义为“IP 地址：端口”的格式。如果端口未定义，则默认情况下该端口设为 9200 使用。可以以逗号分隔的方式来定义多个 Elasticsearch 节点。
- index：指定索引名称，Filebeat 会将事件信息写入到 Elasticsearch 指定的索引中。默认情况下，索引名称是[filebeat-]YYYY.MM.DD。

- proxy_url：指定连接到 Elasticsearch 节点的代理 URL 地址。
- max_retries：指定一个事件发送失败之前重试的最大次数。一旦事件发送失败，这个事件信息就会被删除。

> ℹ有一些 Beats 组件，如 Filebeat，会忽略 max_retries 的设置，并且会一直重试，直到事件信息发送成功。

2. 配置 Logstash 的 output

Logstash 的一些输出设置如以下部分所示：

```
output.logstash:
    #enabled: true
    hosts: ["localhost:5044"]
    #index: "filebeat"
    #loadbalance: true
    #worker: 1
```

下面解释每个属性的含义，这样可以很容易熟悉这些属性：

- hosts：指定要连接的 Logstash 服务器地址。如果端口未定义，则默认情况下该端口设为 5044 使用。可以以逗号分隔的方式来定义多个 Logstash 节点。
- index：指定索引名称，Filebeat 会将事件信息写入到指定的索引中。默认情况下，索引名称是 filebeat。
- loadbalance：在多个 Logstash 服务器主机之间执行负载均衡。如果将其设为 false，网络中同时有多台 Logstash 服务器主机时，则数据将会随机发送给其中一台 Logstash 服务器主机。
- worker：指定要发送事件到 Logstash 的进程数量。当负载平衡被设为 true 时，最好修改这个属性的值。

3. 配置 Logging

Logging 配置的一些设置如以下部分所示：

```
#logging.level: info
#logging.to_syslog: true
logging.to_files: true
logging.files:
    #path: /var/log/filebeat
    #name: filebeat
```

下面解释每个属性的含义，这样可以很容易熟悉这些属性：

- logging.level：在记录日志时指定日志级别。它可以包含诸如 critical、error、warning、info 和 debug 等值。默认情况下它被设置为 info。
- logging.to_syslog：设置为 true 时，用来传输所有输出的日志信息到系统日志中。默认情况下被设置为 false。
- logging.to_files：将所有输出的日志写入到文件。要启用记录日志到文件的功能，其值应指定为 true。
- path：指定存储输出的日志文件的目录位置。
- name：指定存储输出日志的文件名称。

5.5.2 理解 Metricbeat

正如前面所讨论的，Metricbeat 是一个开源的轻量级数据传输工具。它可以读取整个系统中的统计信息，并提供进程级信息，如系统中每个进程的 CPU、内存使用量，以及系统中运行的各种服务的信息。Metricbeat 组件负责收集所有统计数据并发送到多种组件中，如 Elasticsearch、Logstash、Kafka、Redis、File 和 Console。

Metricbeat 通过监控服务器并收集系统以及支持的服务程序中的信息，提供大量的统计数据和指标。

所支持的部分服务如下所示：

- Apache；
- Couchbase；
- Docker；
- Kafka；
- MySQL；
- Nginx；
- PostgreSQL；
- Redis；
- System；
- Zookeeper。

让我们来学习系统模块 System Module，使用 Metricbeat 可以很容易得到这些指标。

5.5.2.1 系统模块

系统模块（system module）包含各种统计数据的指标集（metricset），它们被分为多个不同的部分，如 CPU 级统计数据、文件级统计数据、内存级统计数据、网络级统计数据和进程级统计数据等。

让我们来探讨系统模块的各种指标集。

1. CPU 指标集

这个指标集包含 CPU 的以下性能指标：

- 空闲 CPU 时间的百分比和数量；
- 等待 I/O 磁盘操作所花费的 CPU 时间百分比和数量；
- 处理硬件中断的 CPU 时间百分比和数量；
- 在低优先级进程上花费的 CPU 时间百分比和数量；
- 处理软件中断所花费的 CPU 时间百分比和数量；
- 另一个进程正在由虚拟机管理程序（仅适用于 UNIX 系统）运行服务时，虚拟 CPU 等待时间的百分比和数量；
- 内核空间中 CPU 时间的百分比和数量；
- 用户空间的 CPU 时间百分比和数量。

这个指标集可用于 Windows、Linux、FreeBSD、OpenBSD 和 Darwin OS 等操作系统。

> ℹ核心的指标集在每个核心级别提供了相似的 CPU 信息。

2. 磁盘 I/O 指标集

这个指标集包含磁盘 I/O 的以下指标：

- 执行 I/O 操作所需的总毫秒数；
- 磁盘名称；
- 磁盘成功读取的总字节数（在 Linux 上它相当于读取扇区的数量×512，其中 512 是扇区大小）；
- 成功完成读取的总数；
- 所有读操作所花费的总毫秒数；
- 成功写入磁盘的总字节数（在 Linux 上它相当于读取扇区的数量×512，其中 512 是扇区大小）；
- 成功完成写入的总数；
- 所有写入操作所花费的总毫秒数。

这指标集可以在 Linux、Windows、和 FreeBSD（AMD64）系统上使用。

3. 文件系统指标集

这个指标集包含文件系统的以下统计指标：

- 普通用户可用的磁盘空间；
- 磁盘名称；

- 文件系统中存在的文件节点总数；
- 可用的总磁盘空间；
- 文件系统上存在的所有文件节点总数；
- 磁盘的挂载位置；
- 已经使用的磁盘空间及其百分比。

这个指标集可用于 Linux、Windows、OpenBSD、FreeBSD 和 Darwin OS 等操作系统。

4. FsStat 指标集

这个指标集涵盖了挂载文件系统统计数据的以下指标：

- 当前文件系统计数；
- 当前文件总数；
- 空闲磁盘空间总量；
- 已使用磁盘空间总量；
- 可用磁盘空间总量(空闲＋已使用)。

这个指标集可用于 Linux、Windows、OpenBSD、FreeBSD 和 Darwin OS 等操作系统。

5. 负载指标集

这个指标集包括提供负载统计的以下指标：

- 最后 1 分钟的平均系统负载；
- 最后 15 分钟的平均系统负载；
- 最后 5 分钟的平均系统负载；
- 最后 1 分钟通过内核数归一化的平均系统负载；
- 最后 15 分钟通过内核数归一化的平均系统负载；
- 最后 5 分钟通过内核数归一化的平均系统负载。

这个指标集可用于 Linux、OpenBSD、FreeBSD 和 Darwin OS 等操作系统。

6. 内存指标集

这个指标集涵盖了内存统计数据的以下指标：

- 实际空闲内存容量；
- 实际已使用内存的字节数及其百分比；
- 空闲内存总量；
- 可用的交换空间[①]容量；

① 译者注：当操作系统中的物理内存消耗殆尽时，系统会将内存中不常用的一部分数据换出到磁盘中，以腾出空间来运行频繁使用的程序。这样的技术即为交换空间(swap memory)，在 Windows 系统中称为虚拟内存(virtual memory)。

- 交换空间总量；
- 已使用交换空间字节数及其百分比；
- 内存总容量；
- 已使用内存的字节数及其百分比。

这个指标集可用于 Linux、Windows、OpenBSD、FreeBSD 和 Darwin OS 等操作系统。

> 实际的空闲内存容量取决于不同的操作系统。如 Linux 系统的内存容量由空闲内存、高速缓存和缓冲区组成；OSX 系统的内存容量由空闲内存和交互内存组成；Windows 系统的内存容量等于空闲内存总量。

7. 网络指标集

这个指标集涵盖了网络统计数据的以下指标：

- 接收和发送的字节总数；
- 被删除的传入数据包的总数；
- 被删除的传出数据包总数；
- 接收和发送时的错误总数；
- 接收和发送的数据包总数；
- 接口名称。

这个指标集可用于 Linux、Windows、FreeBSD 和 Darwin OS 等操作系统。

8. 进程指标集

这个指标集涵盖了进程级统计数据的以下指标：

- 命令行中用于启动进程的进程；
- CPU 启动进程的时间；
- 进程所花费的 CPU 时间百分比(类似于 top 命令返回结果中的%CPU 指标)；
- 由进程启动文件描述符 File-Descriptors 后，其数量的软、硬限制；
- 进程启动的文件描述符 File-Descriptors 总数；
- RAM 内存中进程占用的内存百分比和总数；
- 进程使用的共享内存和虚拟内存总容量；
- 进程名称；
- 进程组 ID、父进程 ID 和进程 ID；
- 进程状态；
- 启动进程的用户名(如果用户名不存在，就使用它的 UID)。

这个指标集可用于 Linux、Windows、FreeBSD 和 Darwin OS 等操作系统。

了解 Metricbeat 提供的统计信息之后，下面介绍如何安装和使用 Metricbeat 组件。

5.5.2.2 Metricbeat 的安装

在这一节中，分别在 Ubuntu 和 Windows 系统中完成 Metricbeat 5.1.1 的安装。

要在 Ubuntu 中安装 Metricbeat，请参照以下步骤：

(1) Metricbeat 安装前，请检查系统是 32 位还是 64 位。可以使用下面的命令：

```
uname -m
```

如果命令返回 x86_64，这意味着它是 64 位系统；如果命令返回 i686，这意味着它是 32 位系统。

(2) 使用终端下载 Metricbeat 5.1.1 的 Debian 软件包。

对于 64 位系统：

```
wget https://artifacts.elastic.co/downloads/beats/metricbeat/metricbeat-5.1.1
-amd64.deb
```

对于 32 位系统：

```
wget https://artifacts.elastic.co/downloads/beats/metricbeat/metricbeat-5.1.1
-i386.deb
```

(3) 使用下面的命令来安装 debian 软件包。

对于 64 位系统：

```
sudo dpkg -i metricbeat-5.1.1-amd64.deb
```

对于 32 位系统：

```
sudo dpkg -i metricbeat-5.1.1-i386.deb
```

> Metricbeat 将被安装在/usr/share/metricbeat 目录下，配置文件将保存在/etc/metricbeat 目录下，init 脚本将保存在/etc/init.d/metricbeat 目录下，日志文件将保存在/var/log/messages/metricbeat 目录下。

(4) 配置 Metricbeat 在操作系统启动时自动运行。如果正在使用 SysV init，那么运行下面的命令：

```
sudo update-rc.d metricbeat defaults 95 10
```

上面的命令将打印在屏幕上——添加/etc/init.d/metricbeat 随系统启动的设置。

使用以下命令来检查 Metricbeat 的状态：

```
sudo service metricbeat status
```

使用以下命令将 Metricbeat 作为一个服务运行：

```
sudo service metricbeat start
```

Metricbeat 命令的用法如下：

```
sudo service metricbeat {start|stop|status|restart|force-reload}
```

> 如果将 Metricbeat 作为服务来运行，将运行/etc/metricbeat/metricbeat.yml 配置文件。

> TIP 如果想安装其他版本的 Metricbeat 组件，可以访问 Elastic 团队的下载页面，复制 debian 包的链接并使用 wget 命令来获取安装包。在 Windows 操作系统上安装 Metricbeat 的方法详见 https://github.com/kravigupta/mastering-elastic-stack-code-files/wiki/appendix。

5.5.3　理解 Packetbeat

Packetbeat 是平台中主要的 Beats 组件，它为 Beats 平台的开发铺垫了道路。由于 Packetbeat 超强的流行程度，在 Elastic 收购 Packetbeat 之后，Beats 作为一个概念化的平台出现。正如前面所讨论的，Packetbeat 是开源的轻量级数据包分析器，它用来监视网络上的各种系统和应用程序。Packetbeat 可以在不同的应用服务器之间抓取网络级别的信息。

Packetbeat 获取不同的应用服务器之间的网络信息流通量，然后解析应用服务器使用的协议，将响应信息与请求信息联合对应，这样所有的信息都记录在一个事务中。Packetbeat 可以提供所需的信息，例如应用程序因漏洞而发生的错误，或与性能相关的问题。

此外，它使故障诊断更加容易，并加快了修复过程，因为从存储的信息那里可以很容易分析问题的根本原因。

Packetbeat 支持的协议如下：

- HTTP；
- MySQL；
- MongoDB；

- Redis；
- ICMP；
- DNS；
- PostgreSQL；
- Thrift；
- Memcache。

解析协议和消息之后，Packetbeat 将请求与响应关联起来，这里的响应即为我们熟悉的事务 transaction。如果输出到 Elasticsearch，那么每一个事务都以 JSON 文档的形式存储在 Elasticsearch 中。Packetbeat 使用 Elasticsearch 和 Kibana 来提供网络流通量信息。在这种情况下，甚至可以使用 Filebeat 分析存储在 Logstash 中的多种日志文件，使用它们在同一个系统上进行网络流量分析和日志文件分析。

下面在 Ubuntu 14.04 系统中完成 Packetbeat 5.1.1 的安装。

要在 Ubuntu 中安装 Packetbeat，请参照以下步骤：

(1) Packetbeat 安装前，请检查系统是 32 位还是 64 位。可以使用下面的命令：

```
uname -m
```

如果命令返回的信息是 x86_64，这意味着它是 64 位系统。另外，如果命令返回 i686，这意味着它是一个 32 位系统。

(2) 使用终端下载 Packetbeat 5.1.1 的 Debian 软件包。

对于 64 位系统：

```
wget https://artifacts.elastic.co/ downloads/beats/packetbeat/ packetbeat-5.1.1
-amd64.deb
```

对于 32 位系统：

```
wget https://artifacts.elastic.co/ downloads/beats/packetbeat/ packetbeat-5.1.1
-i386.deb
```

(3) 使用下面的命令来安装 debian 软件包：

对于 64 位系统：

```
sudo dpkg -i packetbeat-5.1.1-amd64.deb
```

对于 32 位系统：

```
sudo dpkg -i packetbeat-5.1.1-i386.deb
```

> Packetbeat 将被安装在/usr/share/packetbeat 目录下，配置文件将保存在/etc/packetbeat 目录下，组件的初始化脚本将保存在/etc/init.d/packetbeat 目录下，日志文件将保存在/var/log/packetbeat 目录下。

（4）配置 Packetbeat 在操作系统启动时自动运行。如果正在使用 SysV init，那么运行下面的命令：

```
sudo update-rc.d packetbeat defaults 95 10
```

上面的命令将打印在屏幕上——添加/etc/init.d/packetbeat 随系统启动的设置。

使用以下命令来检查 Packetbeat 的状态：

```
sudo service packetbeat status
```

使用以下命令将 Packetbeat 作为一个服务运行：

```
sudo service packetbeat start
```

Packetbeat 命令的用法如下：

```
sudo service packetbeat{start|stop|status|restart|force-reload}
```

> 如果将 Packetbeat 作为服务来运行，它将运行/etc/packetbeat/packetbeat.yml 配置文件。

> TIP 如果想安装其他版本的 Packetbeat 组件，可以访问 Elastic 团队的下载页面，复制 debian 包的链接并使用 wget 命令来获取安装包。在 Windows 操作系统上安装 Packetbeat 的方法详见 https://github.com/kravigupta/mastering-elastic-stack-code-files/wiki/Appendix。

5.6　社区开发者开发的 Beats 组件

由于 Elastic Stack 的各类组件都是开源项目的一部分，Beats 已经被开源社区接受，它正积极促进不同类型 Beats 组件的创建。社区开发者遵循 Beats 平台开发了许多新的 Beats 组件。本节了解并探讨社区开发者开发的 Beats 组件，完成安装，并了解为 Beats 组件提供的配置选项。

这是社区开发者开发的 Beats 组件，提供简单而实用的功能。这个 Beats 组件提供了与 Elasticsearch 集群相关的信息。通过使用 API，它可以提供 Elasticsearch 集群、

Elasticsearch 节点和 Elasticsearch 健康程度的统计数据。它同时也需要通过 Elasticsearch 开放的 API 接口来获取 Elasticsearch 集群的多种统计信息。

下面让我们来安装和使用 Elasticbeat 组件。

1. Elasticbeat 组件的安装

在这一节中，将在 Ubuntu 系统中完成 Elasticbeat 的安装。Elasticbeat 是用 Go 语言开发的，这种语言在安装和使用 Elasticbeat 时都需要用到。

要在 Ubuntu 14.04 系统中安装 Elasticbeat，请参照以下步骤：

（1）使用以下命令下载 Go 语言包：

```
wget https://storage.googleapis.com/golang/go1.7.1.linux-amd64.tar.gz
```

（2）提取压缩包中的内容并移动到自定义位置如/usr/local：

```
tar -xvzf go1.7.1.linux-amd64.tar.gz -C /usr/local/
```

使用 sudo 访问权限来执行上述命令。

（3）在～/.bashrc 文件中添加下面与 GO 语言有关的环境变量：

```
export GOROOT="/usr/local/go"
export GOPATH="$ HOME/go"
export PATH="$ GOROOT/bin:$ PATH"
```

使用下面的命令使 bashrc 文件中的配置生效：

```
source ~/.bashrc
```

（4）使用以下命令验证 Go 语言是否已安装：

```
go version
```

该命令将返回 Go 语言的版本：go version go1.7.1 linux/amd64。

（5）转到开发平台的 workspace 文件夹，并使用以下命令创建一个目录：

```
mkdir -p $GOPATH
cd $GOPATH
mkdir -p src/github.com/radoondas
```

（6）访问 Elasticbeat 的远程存储库，并使用以下命令克隆 Elasticbeat 的副本：

```
cd $GOPATH/src/github.com/radoondas
git clone https://github.com/radoondas/elasticbeat.git
```

> ℹ如果系统中没有安装 git，请使用以下命令安装：
>
> ```
> sudo apt-get install -y git
> ```

(7) 通过使用以下命令编译 Elasticbeat：

```
cd $GOPATH/src/github.com/radoondas/elasticbeat
make
```

如果在运行 make 命令后遇到错误，请确保安装的版本高于 1.5。

(8) 使用下面的命令运行 Elasticbeat：

```
./elasticbeat -e -v -d elasticbeat -c elasticbeat.yml
```

如果 Elasticbeat 无法运行，请检查 elasticbeat.yml 配置文件。如果没有设置注释，则应设置好注释，并再次运行。

这个操作将成功运行 Elasticbeat，组件将开始收集有关 Elasticsearch 节点的信息，并依照配置文件中的配置信息，为收集到的数据在 Elasticsearch 中创建索引。

在 Windows 操作系统上安装 Elasticbeat 的方法详见 https://github.com/kravigupta/mastering-elastic-stack-code-files/wiki/appendix。

2. Elasticbeat 组件的配置

下面的配置示例包含以下属性：

```
input:
    period: 10
    urls:
        -http://127.0.0.1:9200/
    stats:
        cluster: true
        nodes: true
        health: true
```

让我们尝试解释每个属性的含义，这样可以很容易熟悉这些属性：

- period：指定 Elasticbeat 组件获取信息的时间间隔(秒)。默认情况下，它的值设为 10，即 Elasticbeat 组件每过 10 秒就会读一次统计数据。
- urls：指定 Elasticsearch 的服务器列表中的 URL，Elasticbeat 组件需要从这些地址读取统计数据。
- cluster：获取与 Elasticsearch 集群相关的统计数据，可以使用 API 中的_cluster/stats 来获取。

- nodes：获取与该节点相关的统计信息，可以使用 Elasticsearch API 中的_nodes/stats/process、jvm、os、fs、thread_pool、transport、http、breaker 和 script 来获取。
- health：获取与集群有关的统计信息，可以使用 Elasticsearch API 中的_cluster/health 来获取。

5.7 Beats 在 Elastic Stack 中的实战

在了解了各种 Beats 组件、它们的功能以及所提供的信息之后，下面将学习如何入门 Beats 组件，以便与 Elastic Stack 联合使用。我们将介绍以下案例的使用流程：

- 使用 Metricbeat 直接发送数据到 Logstash 中，而 Logstash 会处理日志并存入 Elasticsearch，这样，数据最后会在 Kibana 中执行可视化。
- 使用 Elasticbeat 为数据在 Elasticsearch 中创建索引，并使用 Kibana 进行可视化。

在本节中，将了解如何简单处理组件以及如何轻松地将多个组件连接起来。本节的目的不是显示 Logstash 的强大的处理能力，或者 Beats 组件如何优化多个系统间的性能；而是介绍怎样连接多个组件，使数据流能够通过每一个组件，我们可以将 Beats 组件接收到的输入数据与 Kibana 输出的可视化信息联系起来。

5.7.1 用 Logstash 和 Kibana 探索 Metricbeat

正如前面所讨论的，下面将使用 Metricbeat 收集统计数据并发送到 Logstash，而 Logstash 将按要求解析日志并将解析的数据存储在 Elasticsearch 索引中。Kibana 将使用存储在 Elasticsearch 中的数据来执行可视化，并使用一个已创建的 Metricbeat 面板来提供由 Beats Team 创建的常用可视化统计图表。我们尽量简化叙述，以便更好地理解本节中的工作步骤。

利用 Metricbeat 将输出数据存入 Logstash。要向 Logstash 发送输出数据，需要 Metricbeat 配置文件提供的 Logstash 输出配置。创建一个 Logstash 配置文件，接收 Metricbeat 的输入，并将输出数据存入 Elasticsearch。为此，需要下载并安装为 Logstash 提供的 beats Input 插件（如果之前没有安装），这样 Logstash 就可以从 Metricbeat 读取输入数据了。

同时，我们将对 Logstash 的 Output 插件进行配置，为数据在 Elasticsearch 中创建索引；之后，将使用 Elastic 团队提供的 Beats 面板，导入 Kibana 中已保存的搜索结果、可视化统计图表、索引和面板。

ℹ请确保系统中已安装 Metricbeat 组件。

5.7.1.1 步骤1——配置 Metricbeat 发送数据到 Logstash

在这一步骤中，将名为 metricbeat.full.yml 的 Metricbeat 配置文件修改为 metricbeat.yml，此配置文件保存在/etc/metricbeat 目录中。将输出指定为 Logstash 和 Console(控制台)。

指定控制台输出将有助于了解 Beats 组件能否将数据发送到 Logstash。默认情况下，Logstash 配置主机为 localhost，端口为 5044。

配置文件需要修改，注释掉 elasticsearch 的输出配置，并将 Logstash 的输出配置取消注释。同时，控制台输出的配置将与带有良好缩进格式的输出设置一起被取消注释。如果去掉所有的注释以及换行符的文件，配置文件的内容会像下面这样：

```
metricbeat.modules:
-module: system
   metricsets:
     -cpu
     -load
     -core
     -diskio
   -filesystem
   -fsstat
   -memory
   -network
   -process
   enabled: true
   period: 2s
   processes: ['.*']
output.logstash:
   enabled: true
   hosts: ["localhost:5044"]
output.console:
   enabled: true
   pretty: true
logging.level: info
logging.to_files: true
logging.files:
   rotateeverybytes: 10485760 #=10MB
```

使用下面的命令运行 Metricbeat：

```
sudo /usr/share/metricbeat/bin/metricbeat -c /etc/metricbeat/metricbeat.yml
```

如果能够查看日志，那就说明配置是正确的。

上述前端运行的 Metricbeat 命令只应被用来验证数据是否正确。一旦验证，就应该停止前端进程。

> 要在后端运行 Metricbeat，可运行以下命令：sudo service metricbeat start。
> 执行命令检查 Metricbeat 是否能够解析日志：tailf /var/log/metricbeat/metricbeat。

5.7.1.2 步骤 2——创建一个 Logstash 配置文件

在这一步骤中，将创建一个 Logstash 配置文件，读取 Beats 的输入，并将输出数据存入 Elasticsearch。在创建 Logstash 配置文件之前，应该确认 Logstash 中已经有 beats-input 插件。

要验证 Ubuntu 中的 beats-input 插件是否存在，可使用以下命令：

```
/usr/share/logstash/bin/logstash-plugin list | grep beats
```

如果该插件存在，结果中会显示 logstash-input-beats；如果该插件不存在，可使用以下命令安装 beats-input-plugin 插件：

```
/usr/share/logstash/bin/logstash-plugin install logstash-input-beats
```

> bin/plugin 的使用已经被废弃，它已被 bin/logstash-plugin 代替。

要验证 Windows 中的 beats-input-plugin 插件，在命令提示符窗口中使用以下命令：

```
bin\logstash-plugin list
```

如果该插件存在，则在结果中列出所有可用的插件，并检查 logstash-input-beats 插件；

如果该插件不存在，请在命令提示符窗口中使用以下命令安装 beats-input-plugin 插件：

```
bin\logstash-plugin plugin install logstash-input-beats
```

Logstash 的配置文件内容如下所示：

```
input {
    beats {
        port =>5044
    }
}
```

```
output {
    elasticsearch {
        hosts =>"localhost:9200"
        index =>"%{[@metadata][beat]}-%{+YYYY.MM.dd}"
        document_type =>"%{[@metadata][type]}"
    }
    stdout { codec =>rubydebug }
}
```

正如在前面的配置文件中所看到的，我们指定模板使所有 Beats 组件获取的字段被正确存储在 Elasticsearch 中，Beats 的面板也将导入这些数据。

将创建的 Logstash 配置以文件名 metricbeat. conf 保存到/usr/share/logstash 文件夹中。转到 Logstash 文件夹后使用如下命令运行带有配置文件的 Logstash：

```
$ bin/logstash -f metricbeat.conf
```

Logstash 将运行并解析每个输入的配置文件，并将输入存储在 Elasticsearch 中的 metricbeat-YYYY. MM. dd(YYYY. MM. dd 对应于当前年、月和日)索引中。在控制台中可看到日志，其中将显示存储在 Elasticsearch 中的数据及其对应的各种字段。

> 上述前端运行的 Logstash 命令只应被用来验证数据是否正确。一旦验证，就应该停止前端进程。

一旦可以从 Logstash 的日志中查看数据，就可将 metricbeat. conf 文件复制到/etc/logstash/conf. d 目录下。将这两个进程作为一项服务，可以通过以下命令来运行：

```
sudo service metricbeat start
sudo service logstash start
```

该命令将运行 Metricbeat 和 Logstash 进程，最终通过在 Elasticsearch 中创建索引来存储数据。

运行 Logstash 之后验证数据是否成功存储在 Elasticsearch 中的另一种方法如下。

在运行 Logstash 服务和 Metricbeat 之前，执行以下命令：

```
curl -XGET localhost:9200/metricbeat-*
```

上述命令将返回 index_not_found_exception 类型的输出错误。

运行 Logstash 和 Metricbeat 服务后，使用以下命令：

```
curl -XGET localhost:9200/metricbeat-*
```

上述命令将返回索引名称及其元数据的输出数据。

5.7.1.3 步骤 3——下载和加载 Beats 组件示例面板

在此步骤中，将下载由 Elastic 团队提供的 Beats 面板直接监控服务器。它可以提供快速入门的一个例子；也可以根据需要自定义和创建搜索、可视化统计图表和面板。

要在 Kibana 中加载面板，需要运行一个脚本，在 Kibana 中创建索引模式(index pattern)，例如 metricbeat-*。这也将加载已创建的搜索结果样本，可视化统计图表和面板。

要在 Ubuntu 中加载 Beats 面板，请使用以下命令：

```
cd /usr/share/{beat-name}
```

Metricbeat 的完整路径为/usr/share/metricbeat。

运行以下命令下载并加载 beats 组件示例面板：

```
./scripts/import_dashboards
```

该命令将在 Kibana 中加载 Beats 面板。

> 在运行 import dashboard 的脚本之前，请确保 Elasticsearch 正在运行。

默认情况下，执行面板导入脚本时，假定 Elasticsearch 正在默认主机和端口{localhost:9200}上运行。如果在不同的配置下运行 Elasticsearch，可以使用-es 选项来加载：

```
./scripts/import_dashboards -es http://192.168.1.12:9200
```

也可以使用其他配置参数，要查看这些参数请访问：

https://www.elastic.co/guide/en/beats/libbeat/1.2/dashboard-load-options.html

5.7.1.4 步骤 4——查看 Beats 组件示例面板

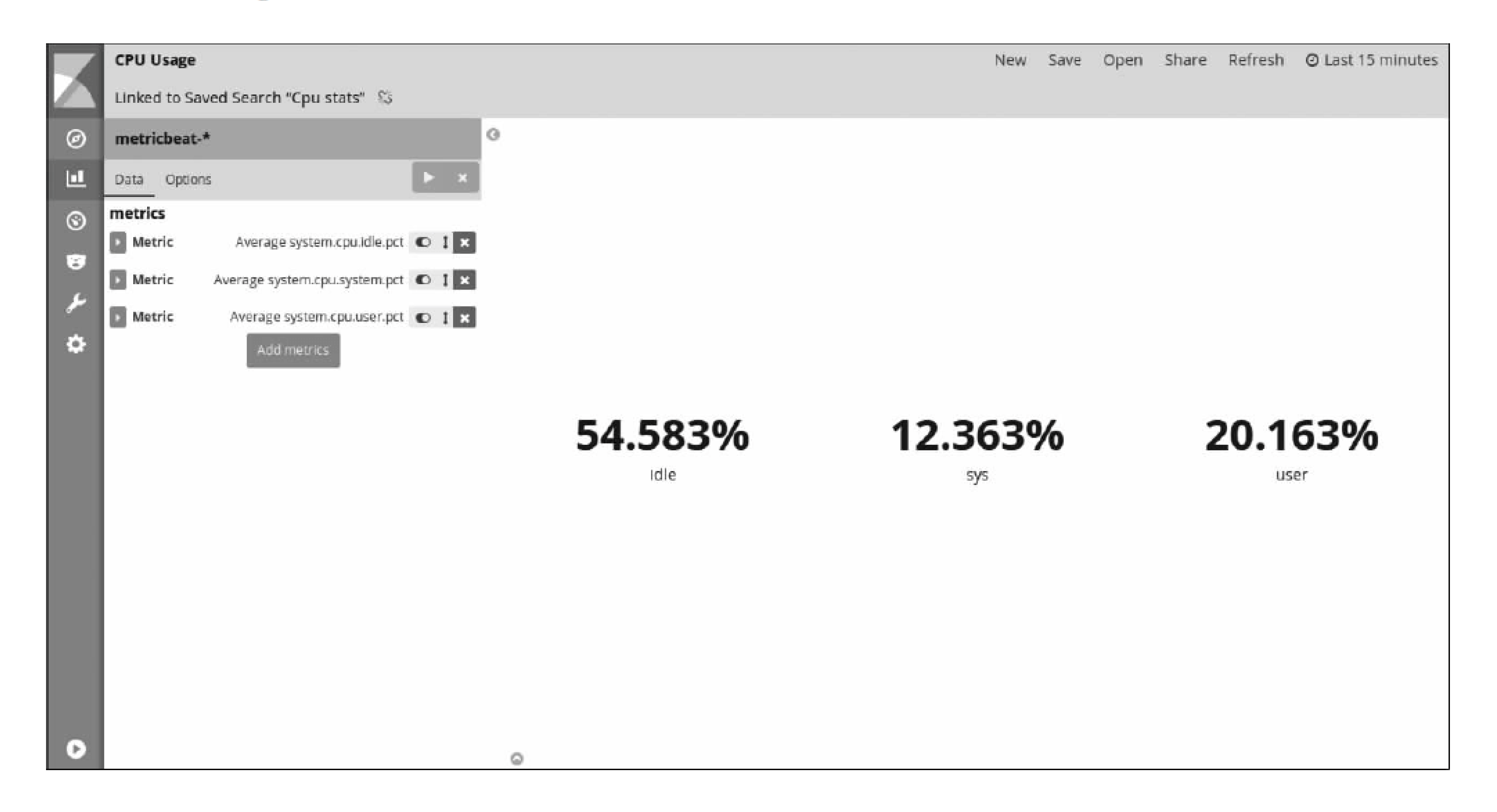
加载示例面板后，即可启动 Kibana，将看到索引模式已创建为 metricbeat-*。也可以单击 Kibana 界面中的 **Management** | **Saved Objects**(**“管理”**|**“已保存的内容”**)导航选项以查看加载到 Kibana 中的所有面板、搜索结果和可视化统计图表。

转到 Kibana 中的 **Visualize**(**可视化**)选项卡，然后从列表中选择可视化方式，或者直接打开 **saved visualization**(**已保存的可视化**)统计图表，该列表将列出所有存储在 Kibana 中的可视化内容。选择 Metricbeat-Dashboard 面板中显示的各种可视化内容，将会看到以下各节所描述的各种可视化统计图表。

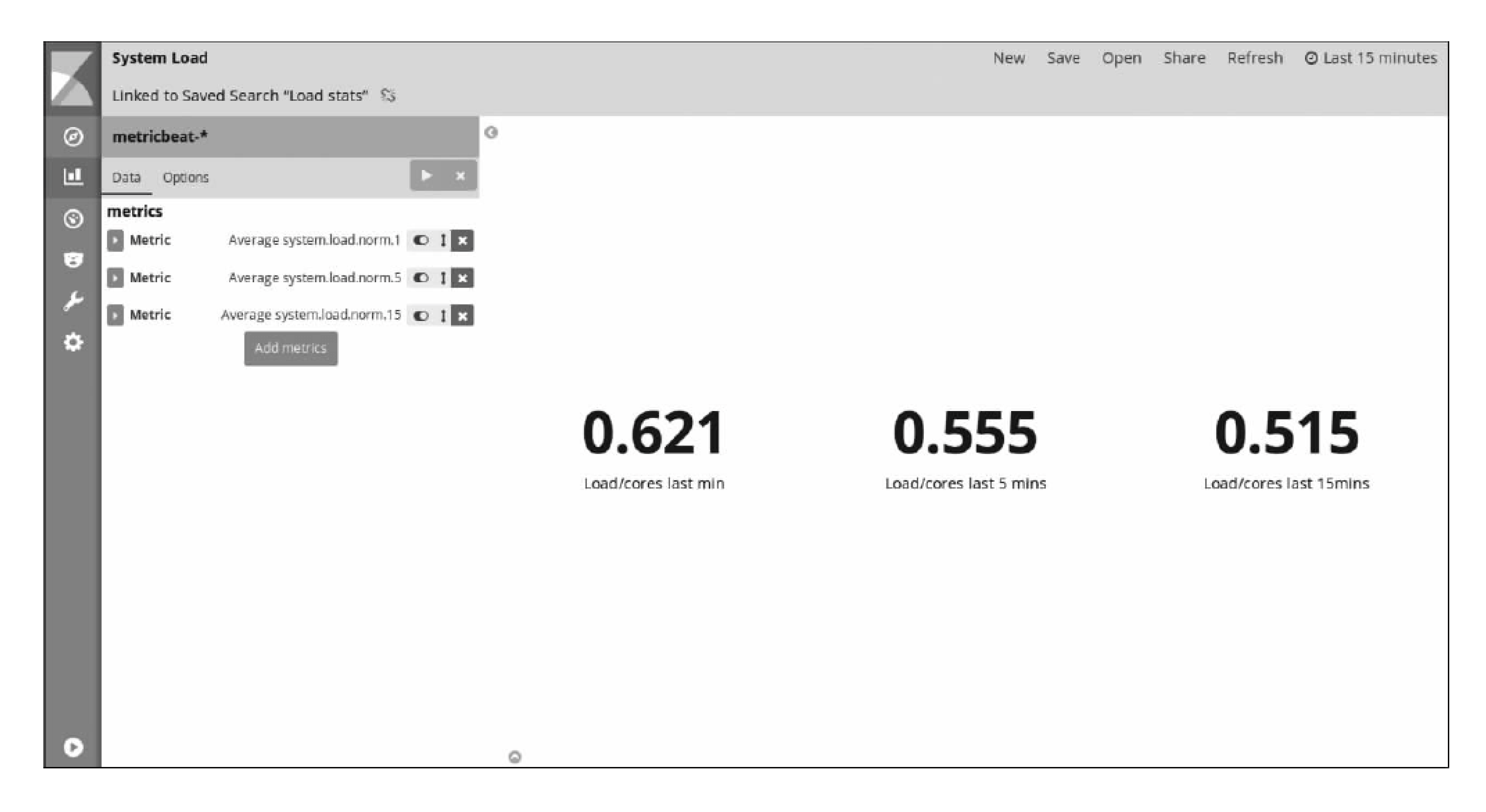
Metricbeat 面板部分会显示基本信息，例如 CPU 使用率、系统负载和 CPU 总占用率在一段时间内的变化情况。

让我们逐个看一下这些可视化内容：

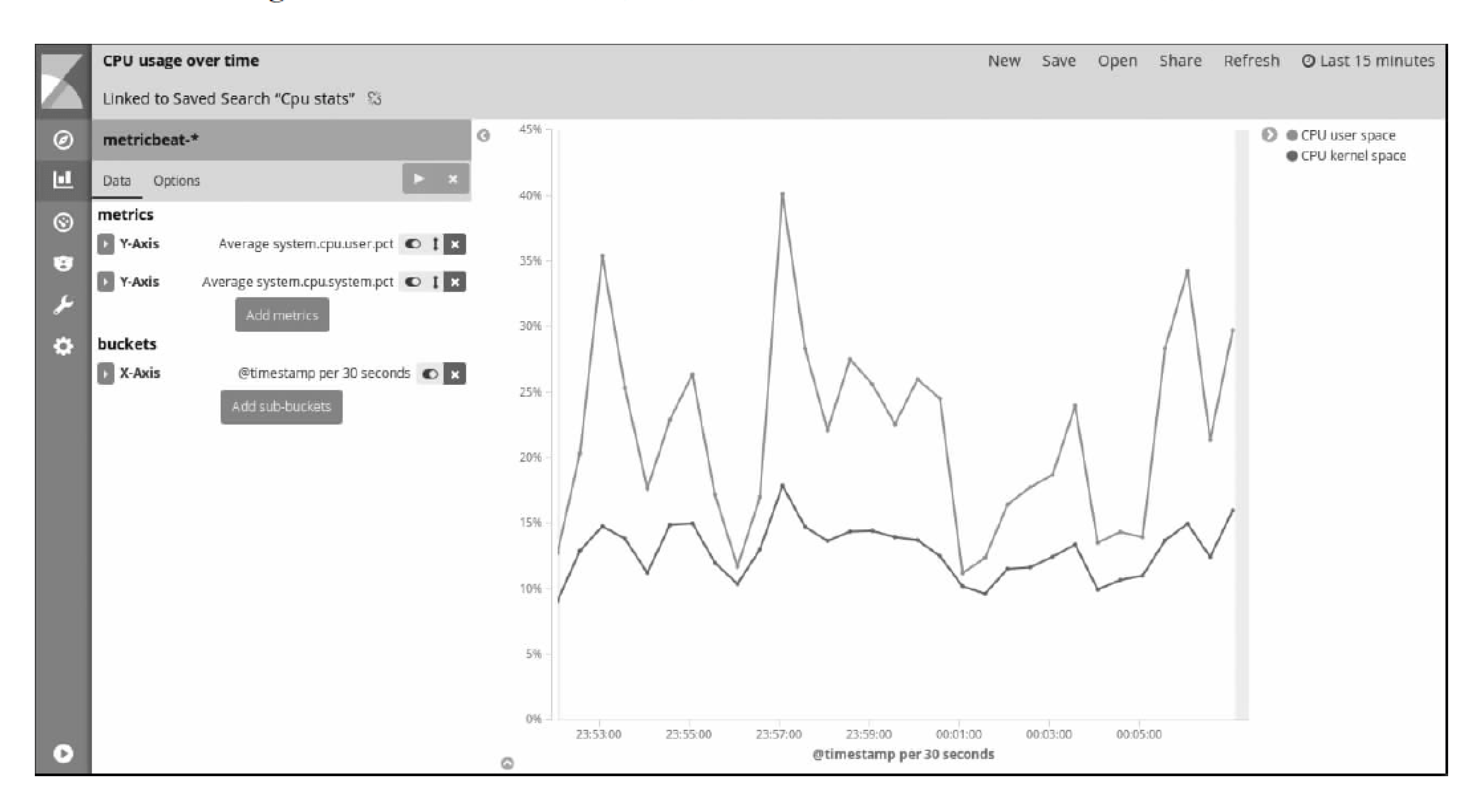
- **CPU Usage**：以下屏幕截图展示了 **CPU 使用率**的界面。



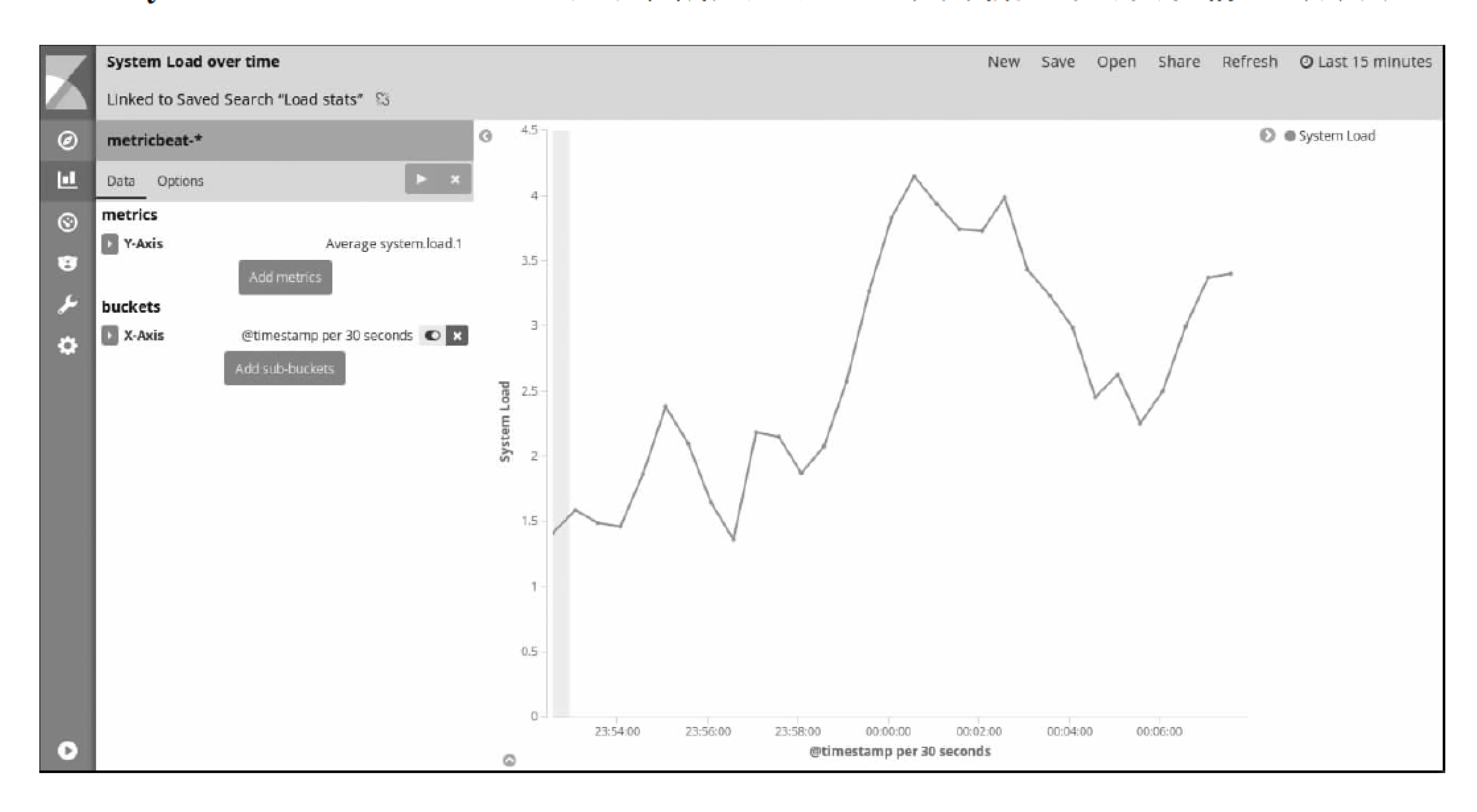
- **System Load**：下面的屏幕截图展示了**系统负载**界面。



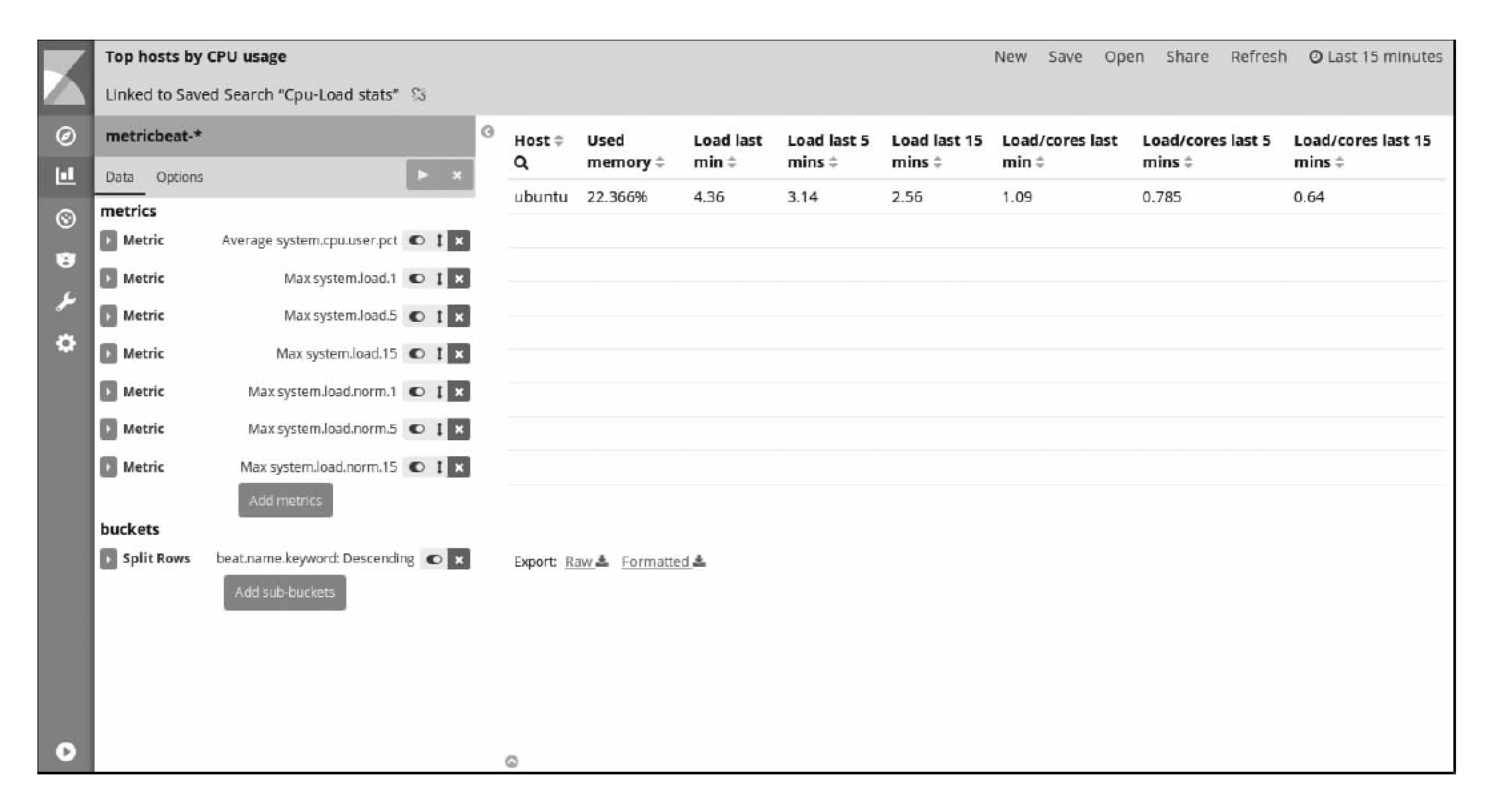
- **CPU Usage over time**：以下屏幕截图展示了 **CPU 使用率随时间变化的界面**。



- **System Load over time**：以下屏幕截图展示了**系统负载随时间变化情况**的界面。

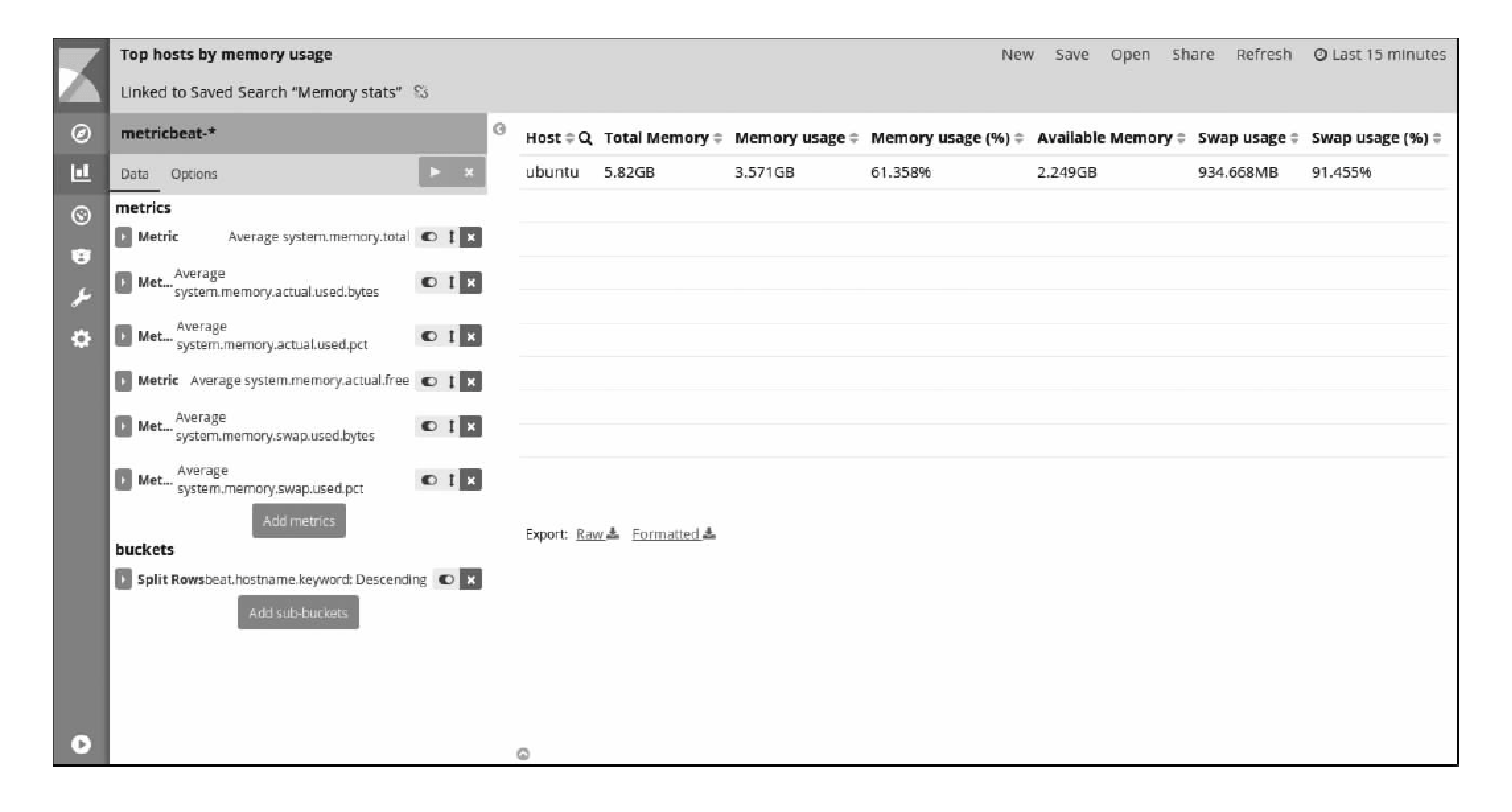


- **Top hosts by CPU usage**：以下屏幕截图展示了 **CPU 占用率最高的主机信息**的界面。

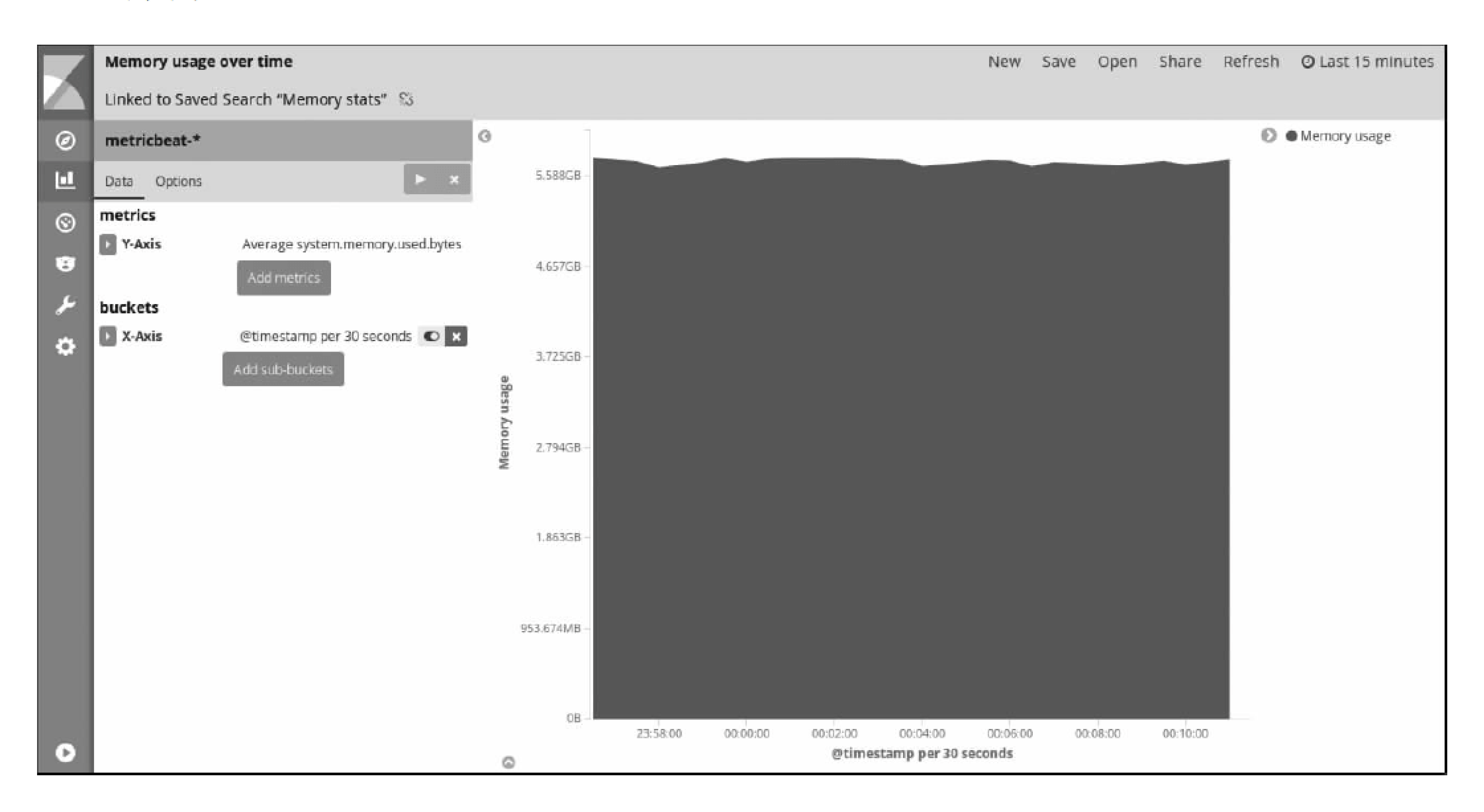


下面看看 Kibana 展示出的其他可视化内容。

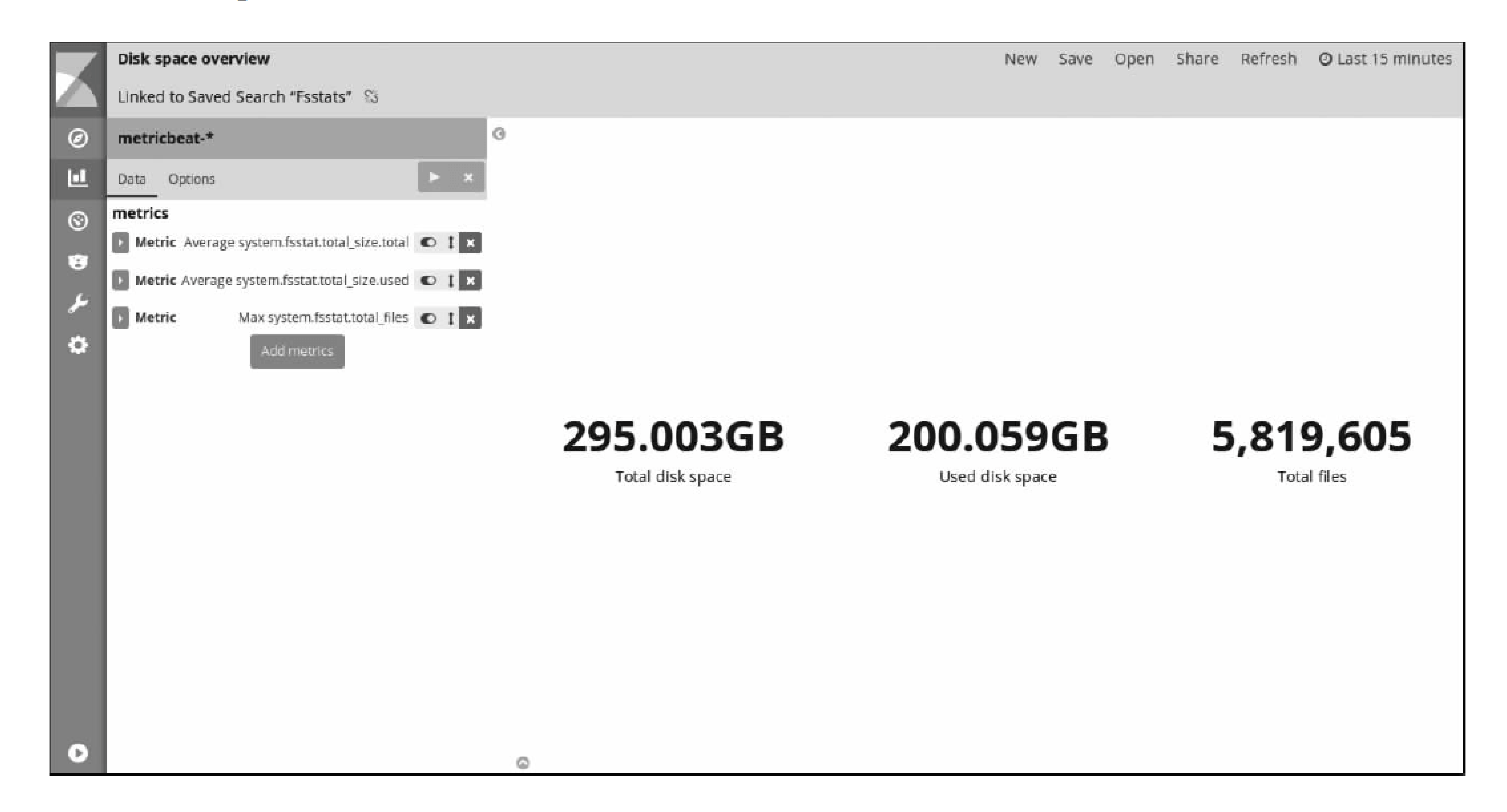
- **Top hosts by Memory usage**：以下屏幕截图展示了**内存占用率最高的主机信息**的界面。



- **Memory usage over time**：以下屏幕截图展示了**内存占用率随时间变化情况**的界面。



- **Disk space overview**：以下屏幕截图展示了**磁盘空间概况**的界面。



5.7.2 用 Elasticsearch 和 Kibana 探索 Elasticbeat

下面将使用 Elasticbeat 收集信息，并将输出的数据发送到 Elasticsearch，为数据创建索引。使用 Elasticsearch 中的数据，Kibana 可以使用 Elasticbeat 社区创建的示例面板对数据执行可视化，并提供查看数据的访问权限。让我们尝试简化叙述，以便更好地了解本节中的工作步骤。

使用 Elasticbeat 并将其输出存储在 Elasticsearch 中。要将输出发送到 Elasticsearch，需配置 Elasticbeat 配置文件中提供的输出配置。之后，将下载由 Elasticbeat 社区提供的 Beats 面板，用于在 Kibana 中导入已保存的搜索结果、可视化内容、索引和面板。

> ℹ请确认系统中已安装 Elasticbeat 组件。

5.7.2.1 步骤 1——配置 Elasticbeat 发送数据到 Elasticsearch

在此步骤中，将修改名为 elasticbeat.yml 的 Elasticbeat 配置文件，该文件将保存在 $GOPATH/src/github.com/radoondas/elasticbeat 中，把输出指定为 Elasticsearch。默认情况下，Elasticsearch 配置主机为 localhost，端口为 9200。

从配置文件中去掉所有的注释以及换行符，配置文件将如下所示：

```
input:
    period: 1
    urls:
        -http://127.0.0.1:9200
    stats:
        cluster: true
        nodes: true
        health: true
output:
    elasticsearch:
    hosts: ["localhost:9200"]
shipper:
logging:
files:
    rotateeverybytes: 10485760 #=10MB
```

执行以下命令转到指定的目录：

```
cd $GOPATH/src/github.com/radoondas/elasticbeat
```

执行以下命令运行 Elasticbeat：

```
./elasticbeat -e -v -d elasticbeat -c elasticbeat.yml
```

此时可以查看 Elasticbeat 组件每隔 1 秒获取的集群统计信息、节点统计信息和集群健康状况的信息。

5.7.2.2 步骤 2——下载和加载 Elasticbeat 面板

在此步骤中，将下载 Elasticbeat 组件的示例面板，提供快速入门的一个例子。也可以根据需要自定义和创建搜索，可视化统计图表和面板。

要在 Ubuntu 中加载 Elasticbeat 面板，可以使用以下命令：

```
cd $GOPATH/src/github.com/radoondas/elasticbeat/kibana
curl -L -O https://github.com/elastic/beats-dashboards/blob/master/load.sh
chmod u+x load.sh
./load.sh
```

执行以上命令将在 Kibana 中加载 Elasticbeat 面板。

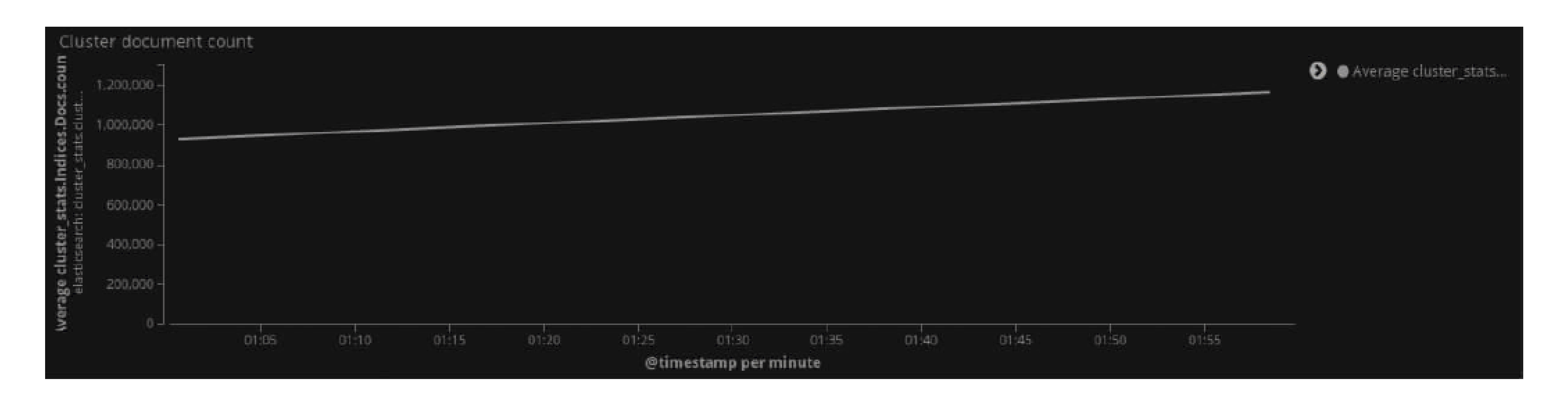
> ℹ运行加载脚本之前，请确保 Elasticsearch 正在运行。

5.7.2.3 步骤 3——查看 Beats 组件的示例面板

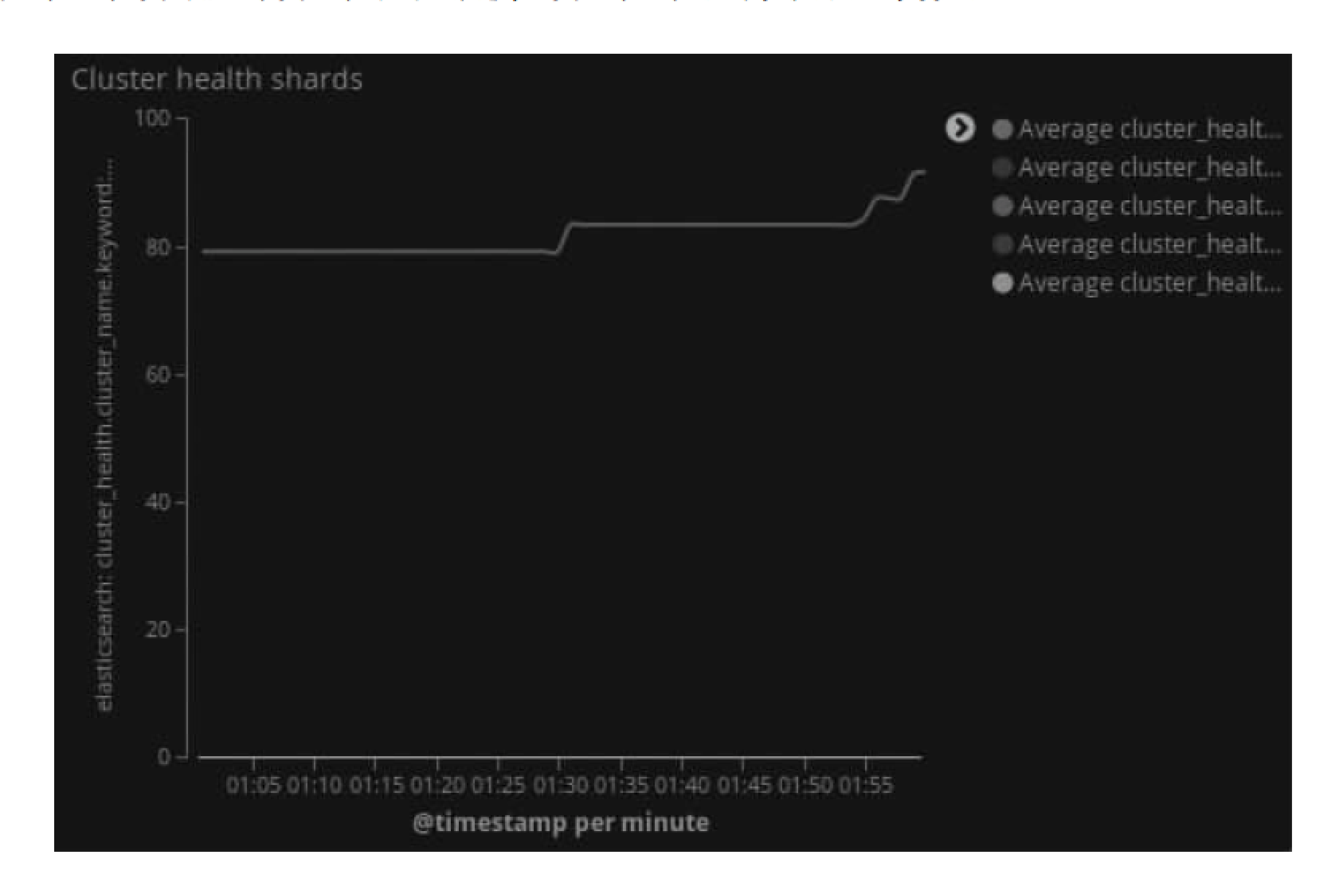
加载示例搜索结果、可视化内容和面板之后，即可启动 Kibana，将看到 Kibana 中已经创建了 elasticbeat-* 的索引模式。也可以转到 **Management** | **Saved Objects（管理 | 已保存的内容）**，查看加载到 Kibana 中的面板、搜索结果和可视化内容。

转到 Kibana 中的 **Visualize** 选项卡，然后从列表中选择可视化方式，或者直接打开 **saved visualization（已保存的可视化）**统计图表。该列表将列出所有存储在 Kibana 中的可视化内容。选择 Metricbeat-Dashboard 面板中显示的各种可视化内容，将会看到以下所描述的各种可视化统计图表。

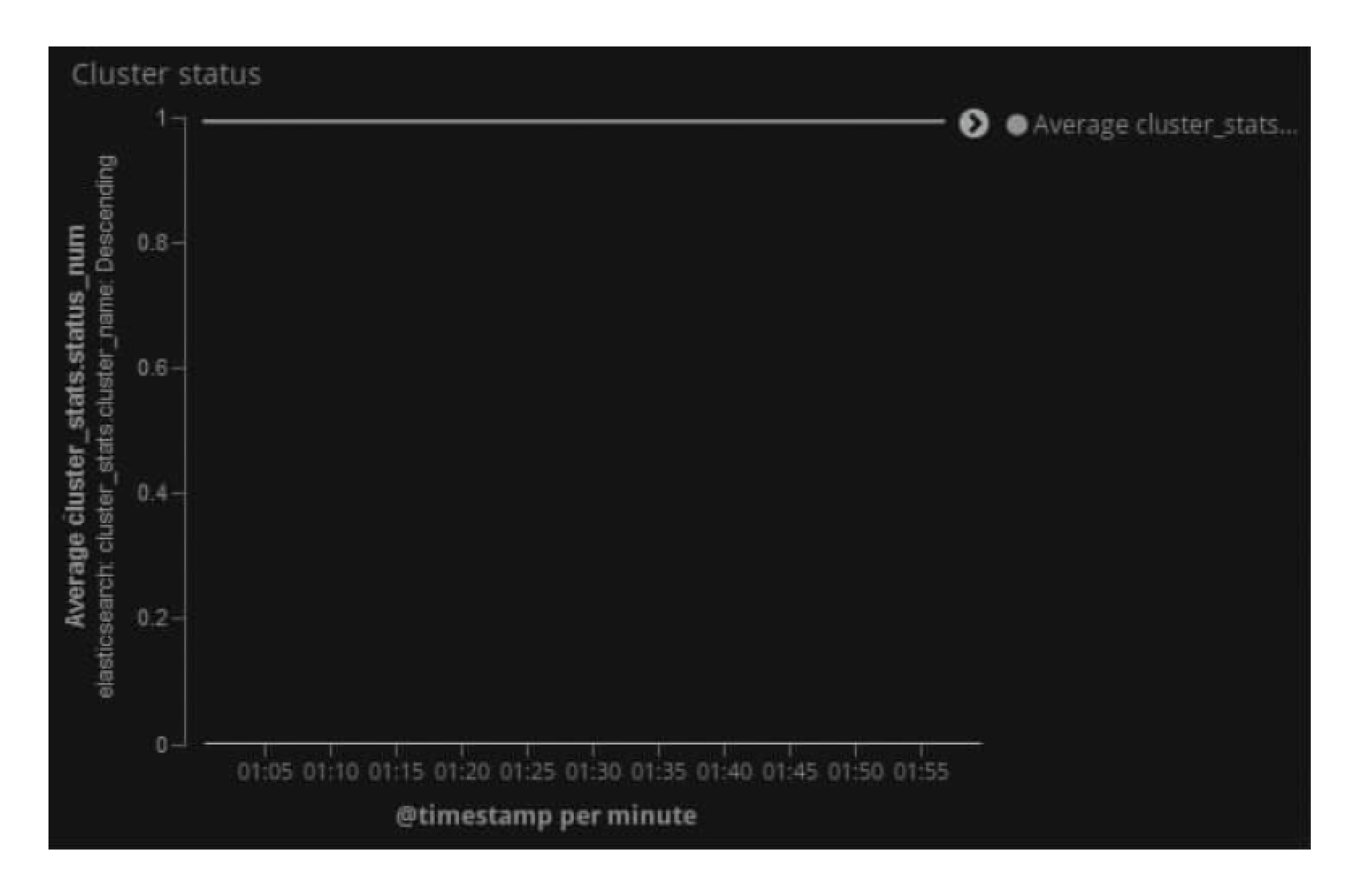
- **Cluster document count**：以下可视化内容展示了在一段时间内，每个集群中存在的文档计数。



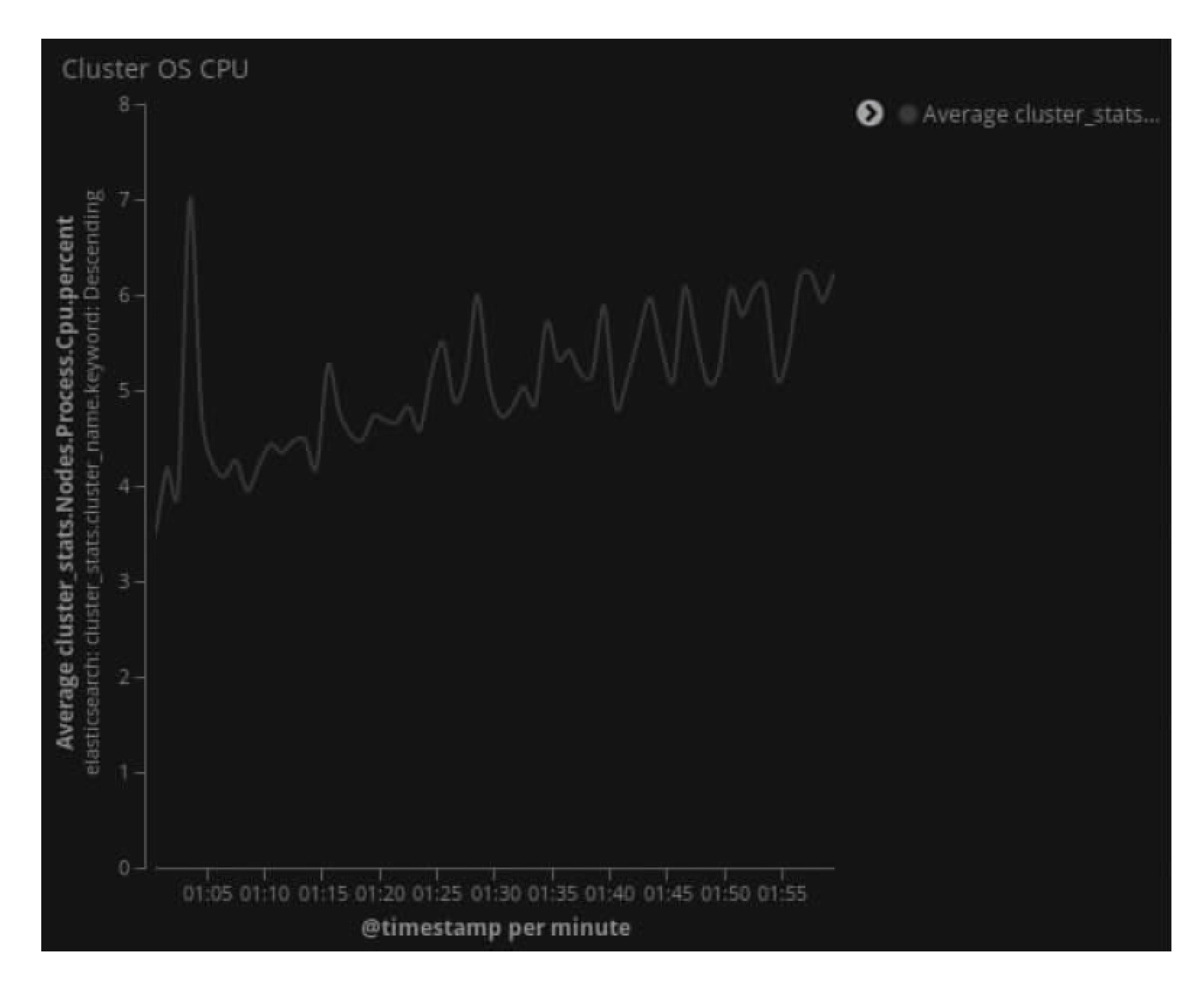
- **Cluster health shards**：以下可视化内容展示了每个集群中初始化分片、活动主分片、主分片、未分配的分片和重定位分片的运行状况计数。



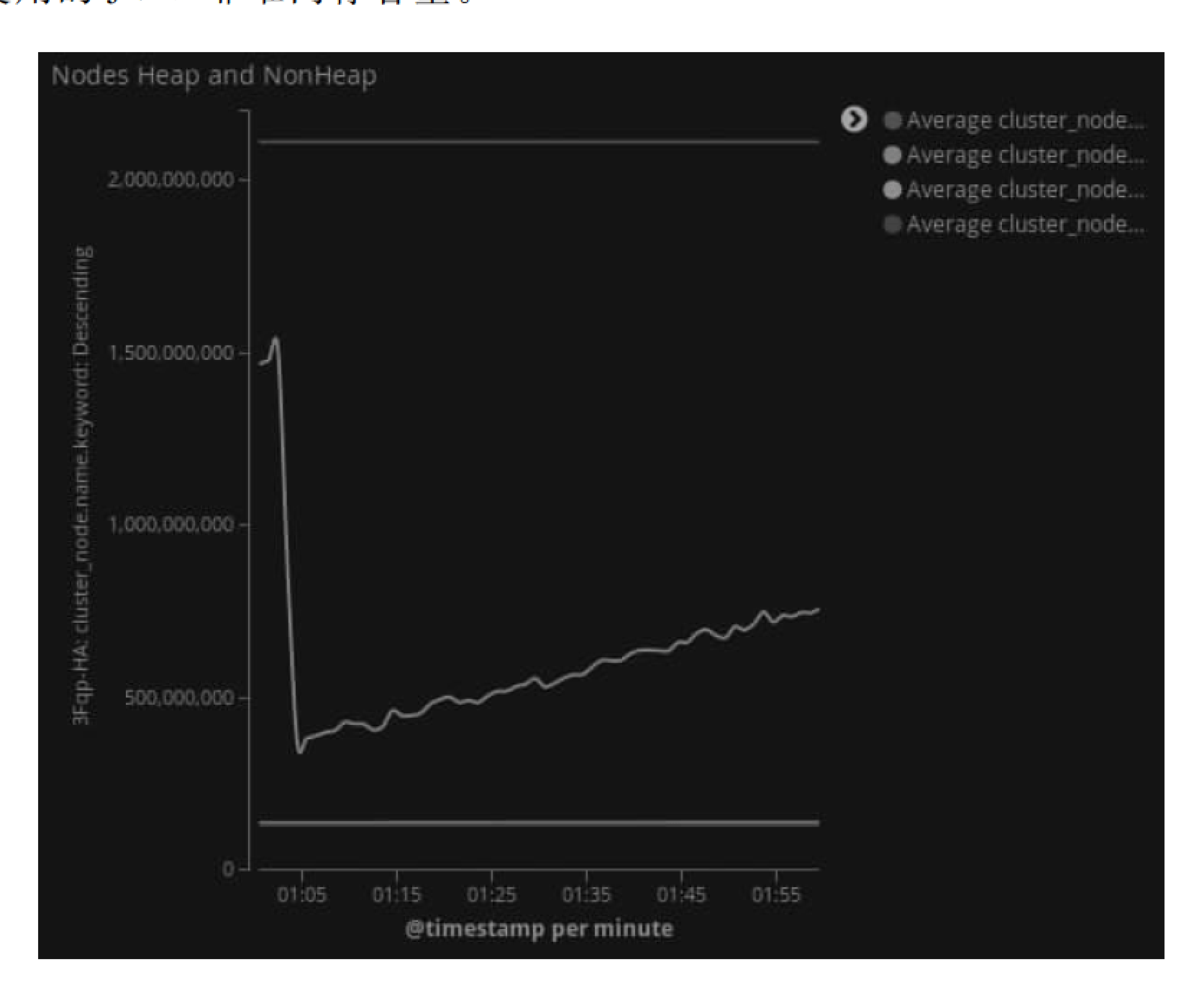
- **Cluster status**：以下可视化内容展示了在一段时间内，每个集群中存在的集群状态节点的平均数量。



- **Cluster OS CPU**：以下可视化内容展示了在一段时间内，每个集群中的节点所占用CPU的平均百分比。



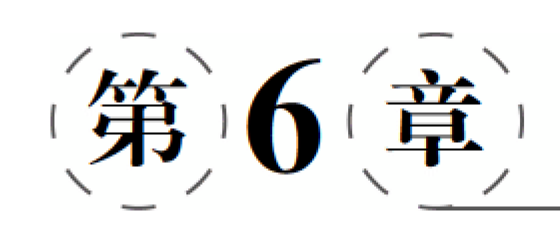
- **Nodes Heap and NonHeap**：以下可视化内容展示了在一段时间内，每个集群中节点所使用的平均 JVM 堆内存容量、已使用的 JVM 堆内存容量、JVM 非堆内存容量和已使用的 JVM 非堆内存容量。



5.8 本章小结

在本章中，我们了解了最新添加到 Elastic Stack 中的 Beats 组件。本章内容涵盖了 Beats 组件被开发或出现的基本前提，以及它与 Logstash 的区别。我们探讨了 Beats 组件在 Elastic Stack 中的作用以及 Beats 组件对以前的 ELK Stack 扩展的重要性；之后，还探讨了不同类型的 Beats 以及它们各自的功能、角色和配置选项；此外还详细介绍了如何安装和配置 Beats 组件。在本章末尾，介绍了 Beats 组件在 Elastic Stack 中使用的示例。

在下一章中，将介绍在实时的生产环境中解决一个特定的问题。我们将探讨 Elastic Stack 如何解决所述的问题，并展示 Elastic Stack 的强大功能。



Elastic Stack 实战

Elastic Stack 服务于许多组织，为这些组织解决实际问题。一个常见的场景是使用 Elastic Stack 组件为企业内网(Intranet)提供解决方案。在本章中，将选取这样一个内网应用场景。迄今为止，我们已经学习了关于设置 Elasticsearch、Logstash、Beats 和 Kibana 协同工作的方法。下面将在用例中使用所有这些组件。

用例是一个简单的应用实现，包括一个门户网站(portal)服务器、一个 Web 服务器、一个搜索引擎和数据库服务器，将分析这些服务器生成的所有类型的日志和节点统计数据。

本章主要内容如下：

- 理解问题场景；
- 准备 Elastic Stack 数据管道 Pipeline；
- 配置 Elastic Stack 组件；
- 设置 Kibana 面板。

6.1 理解问题场景

Elastic Stack 的一个非常流行的用例就是进行日志管理和分析。Elastic Stack 协助了许多此类场景的部署。在下面的例子中，将考虑一个公司的内网。这个内网包括几个模块，用来帮助员工处理公司的事件和流程。这些模块如下：

- **员工信息管理系统**：注册用户并完成信息提供。
- **培训**：设置培训、培训注册、记录考勤等。
- **绩效管理系统**：管理所有员工的评估周期和评估内容。
- **休假管理**：员工使用这个模块申请、批准或拒绝休假。这个模块还可有默认的休假类型及统计、假期列表等。
- **博客**：由公司员工共享知识的内部博客。
- **论坛**：包含了技术类别的内部论坛，员工可以在这里查询并找到答案。
- **Wiki**：公司的 Wiki，列出了项目、流程、策略等可共享的信息。

- **文档库**：集中保存公司所有必需的文档，并为不同的文档设置相关权限。
- **其他几个模块**：这些模块有助于向员工发送通知、生日、周年纪念等内容。

通常，像这样的模块是使用企业门户服务器开发和设置的。对于上述需求，使用了 **Liferay portal** 服务器，它是 portal 的众多行业领导者之一。更多关于 Liferay 的细节可以通过 https://www.liferay.com/搜集到。Liferay 已经提供了许多 **portlet**（模块），公司门户网站可以直接使用或者进行极少更改后使用。我们已经使用了博客、论坛、Wiki、文档和媒体以及与定制开发的模块协作的网络内容模块，即员工信息、休假管理、培训、绩效管理等。

这些都是关于企业门户服务器的。门户部署通常搭建成集群，并且由网络服务器、负载均衡器、搜索引擎、数据库服务器等支持。在下一部分中，我们将介绍如何设置体系结构，从而对这些服务器的优势加以利用。我们选择的 Liferay 版本是 v6.2，这一版本是最新且稳定的版本。Liferay 7 已经推出，但它需要更多的时间才能成为稳定和无故障的版本。

所有这些服务器就绪后，将会有日志分布在这些机器上（服务器程序应安装在不同机器上），这些日志会有不同的格式。对于任何需要注意的事件（错误或缺陷），我们需要从不同的机器读取和同步日志，并对错误或缺陷进行故障排除。

当然，应该收集和存储的信息不仅是应用程序和服务器日志，也应该收集和存储与机器相关的信息。这些日志可以是事件日志、运行时（runtime）RAM、进程或者处理器信息。

使用 Elastic Stack 进行日志管理和分析，可将日志以公共的格式集中在一起，借助它们分析收集的数据。

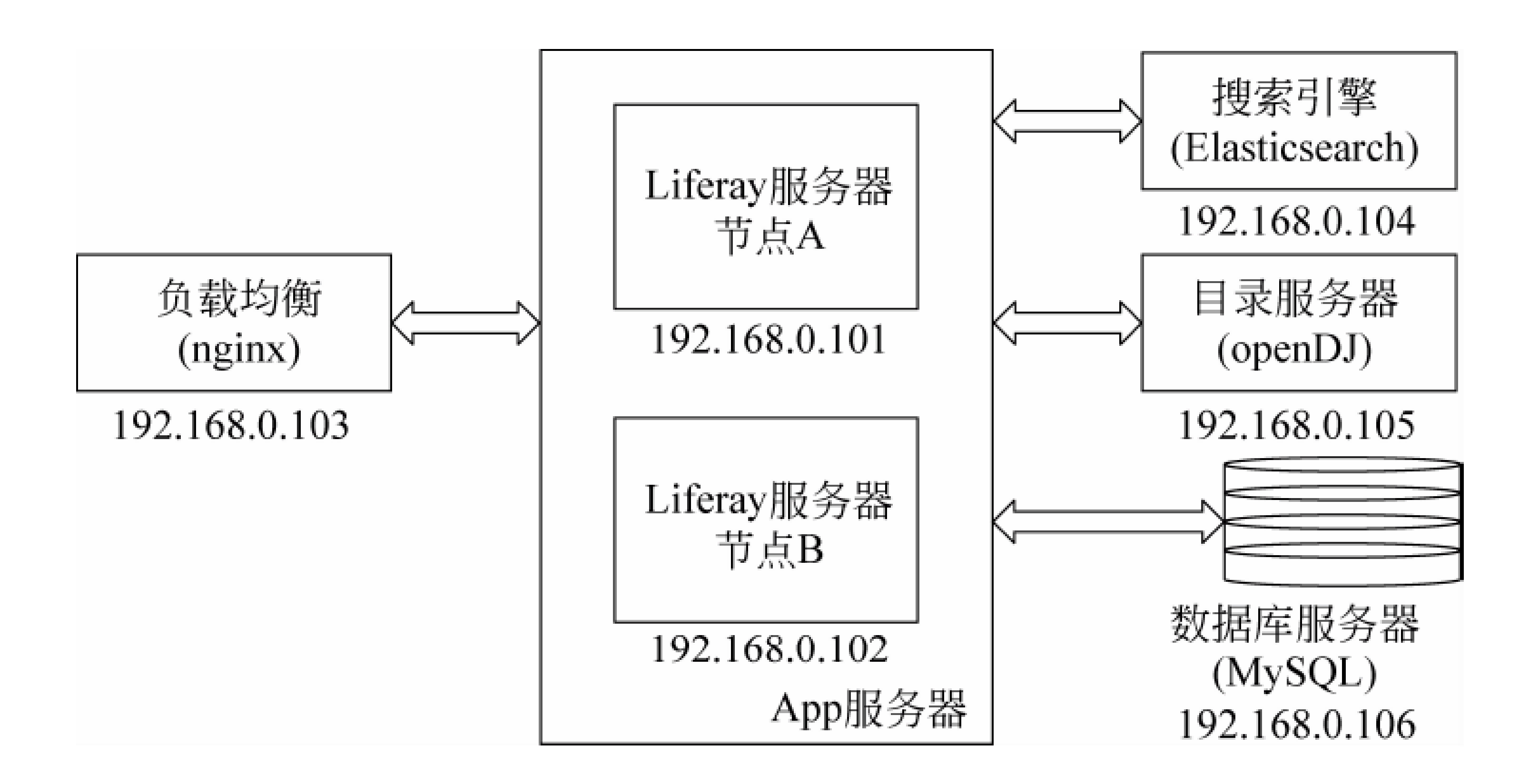
门户服务器总能接收到大量数据。如果这些数据仅负载到一台服务器上，将会导致性能上的问题。我们希望使用两台门户服务器和一个负载均衡器，还有一些其他的工具也可以对改善体系结构做出贡献：

- **Liferay**：承载所有模块的门户服务器平台；
- **OpenDJ**：LDAP 服务器；
- **MySQL**：数据库服务器；
- **Nginx**：负载均衡器；
- **Elasticsearch**：Liferay 门户网站使用 Elasticsearch 存储索引，并且所有搜索请求都将针对此服务器而展开。

下面的示意图展示了这个应用程序的架构和节点的 IP 地址：

从网络架构示意图中可以看出，使用 **Nginx** 作为负载均衡器来平衡负载的主应用服务器有两个节点。**Nginx** 的负载平衡脚本如下：

```
upstream app_liferay_mes {
```




```
    server 192.168.0.101:8080 ; #Liferay Tomcat Node 1
    server 192.168.0.102:8080 ; #Liferay Tomcat Node 2
    ip_hash;
}
server {
    listen 80;
    server_name lportal.com;
    access_log /var/log/nginx/lportal.local.log;
    location / {
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host:$server_port;
        proxy_set_header X-NginX-Proxy true;
        proxy_pass http://app_liferay_mes/;
        proxy_read_timeout 180s;
        proxy_connect_timeout 10s;
    }
}
```

> Liferay 的两个节点都运行在端口 8080 上，并且使用 lportal.com 域连接到 Nginx 的两个节点。

我们添加了一个上游代理。对于负载均衡，选择使用 ip_hash 机制，这个机制使用客户机的 IP 来确定发送请求的节点。当然，还有其他机制：轮转和最少连接机制。使用 ip_hash 将确保请求发送到同一服务器上，并且未来不会出现会话相关的问题。

所有的服务器安装在不同的节点上。由于我们的应用服务器会处理大量的搜索查询，所以在一台单独的服务器上部署搜索引擎 Elasticsearch。默认情况下，Liferay v6.2 和更早的版本都是用 Lucene 来为内容创建索引。对于 Elasticsearch 的支持，可以使用一款名为

Elasticray（https://web.liferay.com/marketplace/-/mp/application/41044606）的插件。这个插件被设置使用 Elasticsearch v1.4，这是与 Liferay 一起使用的版本。

在默认情况下，Liferay v7 使用 Elasticsearch 提供的搜索引擎功能，并嵌入了 Elasticsearch 2.2。也可以使用独立的服务器来运行 Elasticsearch，但服务器版本应该选择 2.2。

所有的用户均使用目录服务器进行管理，OpenDJ（正式名称为 OpenDS）是个很好的选择。对于最后一个节点，即数据库服务器，我们使用 MySQL 数据库服务器。

下表列出了部分系统的所有节点以及它们的版本。

服务器节点	版　本
Nginx 负载均衡器	Nginx/1.10.1
Liferay 服务器节点	6.2
MySQL 数据服务器	5.7.10
OpenDJ LDAP 服务器	3.0.0
Elasticsearch 服务器——针对 Liferay	1.4.0

这些组件的安装和搭建，以及针对这些组件的了解都超出了本书的范围。假设读者对类似软件的搭建都是熟悉的。针对你的用例，你的组件中可能不会用到 Liferay，也可能会用到其他提供类似功能的工具。我们的主要目的是了解 Elastic Stack 如何实现内容存储、可视化和分析。对于任何企业内网门户的搭建，都需要类似的工具集。我们需要一个负载均衡器、数据库、应用服务器、身份验证和用户存储。我们想要讨论的是理解 Elastic Stack 的哪些组件可以用来获取数据，为我们做必要的处理，这样就可以对数据执行可视化，帮助我们采取必要的操作和决策。

为了帮助你完成设置，我们在 https://github.com/kravigupta/mastering-elastic-stack-code-files/wiki/LiferaySetup 创建了一个 wiki；这个 wiki 页面包含了搭建 Liferay 的步骤。

6.2 准备 Elastic Stack 管道

我们有一个需要像前面章节那样进行数据和日志分析的场景，并希望使用 Elastic Stack 组件帮助我们进行分析。这些组件将安装在不同的节点上，并将数据提交到一个中央 Elasticsearch 节点或者一个 Elasticsearch 集群中。为了搭建这样的功能，需要更新我们的体系结构，使其包含 Elastic Stack 组件，如 Logstash、Beats、Elasticsearch 和 Kibana。

6.2.1 要获取什么数据?

在开始更新体系结构之前,需要了解希望获取什么数据,并且这些数据将怎样帮助我们。以下是需要获取的数据:

- 由以下组件生成的日志数据:
  - Liferay、MySQL;
  - Nginx;
  - OpenDJ;
  - 被 Liferay 使用的 Elasticsearch 节点。
- 每个节点的系统统计数据:
  - Liferay、MySQL、Nginx、OpenDJ,以及 Elasticsearch 的所有节点。
- 每个节点的网络流量:
  - 包括 HTTP、MySQL 协议等。

6.2.2 更新体系结构

需要添加合适的 Beats、Elasticsearch、Kibana 和 Logstash 更新系统体系架构。其中应设置一个 Elasticsearch 节点或集群来接收所有通过 Beats/Logstash 读取的数据,以及一台 Kibana 服务器,对记录的数据进行分析。最重要的部分是选择希望安装的 Beats 组件,并选择在哪台机器或者服务器上安装。

我们来分析一下每台服务器,得出选择 Beats 组件的结论:

- **Nginx**:需要获取有关访问和报错的日志、系统运行状态,以及 CPU/内存的性能指标。对于日志,需要 Filebeat;对于 CPU/内存性能指标,可以使用 Metricbeat。我们还希望使用 HTTP 协议的 Packetbeat 来嗅探网络数据包。
- **Liferay 节点**:这些是主要服务器,它将服务于用户的实际请求。需要获取应用服务器、内存和 CPU 使用情况的日志,Filebeat 和 Metricbeat 将会帮助我们。
- **OpenDJ**:这是 LDAP 服务器,它负责身份验证。对于日志和其他与系统相关的指标,也需要 Filebeat 和 Metricbeat。
- **Elasticsearch server for Liferay**:这种 Elasticsearch 致力于提供 Liferay 的搜索功能,它的版本是 1.4.0。我们想要获取内存和 CPU 统计数据和日志,因此需要使用 Metricbeat 和 Filebeat。
- **MySQL 服务器**:使用 Packetbeat 获取与 MySQL 相关的特定数据包。和其他服务器一样,也可以使用 Filebeat 和 Metricbeat 来捕获日志和系统数据。

Elasticsearch server for Liferay 与 Elastic Stack 中的 Elasticsearch 是不同的。一个不

同之处是，Elastic Stack 中的 Elasticsearch 可以存储已获取的数据，并通过 Beats 组件和 Logstash 进行处理。另一个不同之处在于版本，Elastic Stack 的版本是 5.1.1，而另一个则是由 Liferay 插件支持的 1.4.0 版本。

下面的表中指定了不同的服务器在获取相关数据时，所安装的 Beats 组件。

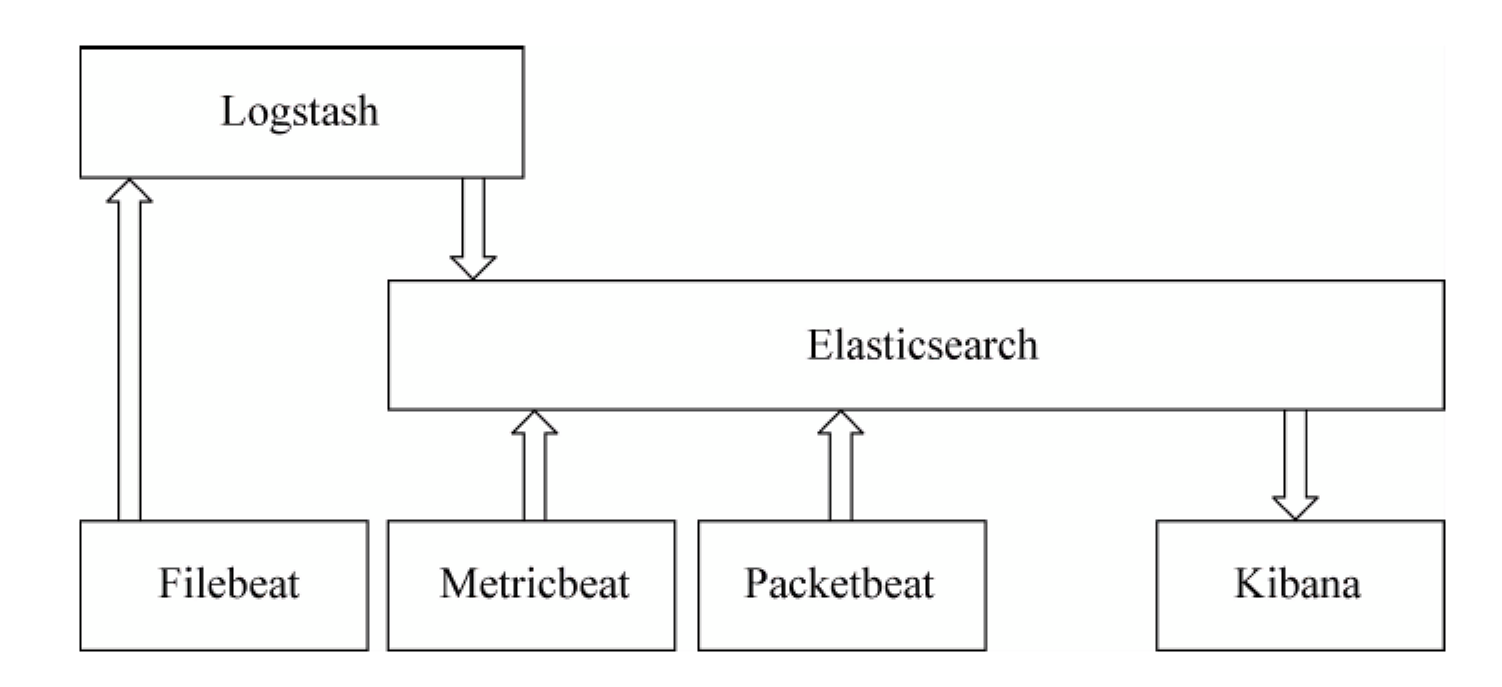
服务器节点	安装的 Beats 组件
Nginx 负载均衡器	Filebeat、Metricbeat 和 Packetbeat
Liferay 服务器节点	Filebeat 和 Metricbeat
MySQL 数据服务器	Filebeat、Metricbeat 和 Packetbeat
OpenDJ LDAP 服务器	Filebeat 和 Metricbeat
Elasticsearch 服务器——针对 Liferay	Filebeat 和 Metricbeat

由于必须读取所有服务器的日志，而日志是存储在文件中的（大部分是.log 文件），所以需要用 Filebeat 来读取日志。对于与网络相关的流量，需要用 Packetbeat；对于系统中的统计数据，可以使用 Metricbeat。

Filebeat 读取的数据将被发送到 Logstash，以便进行一些处理，之后，Logstash 将数据发送到 Elasticsearch。Logstash 会解析日志，并将完整的日志信息分解为多个字段。例如，Nginx 访问日志包含访问者的 IP、referrer、user agent 以及更多的字段。如果不把它分解成多个字段，一定会错失很多高质量的分析结果。对于日志消息的额外处理和解析，使用 Logstash 是首选，因为它提供了灵活性。对于 Metricbeat 和 Packetbeat，将使用 Elasticsearch 作为输出。

6.3 配置 Elastic Stack 组件

在这一节中，我们将配置所有用来获取数据的工具，使用的组件有 Elasticsearch、Logstash、Kibana、Filebeat、Metricbeat 和 Packetbeat。其中的数据管道如下图所示。



所有的组件都是同一版本，即 5.1.1。使用 Filebeat 读取日志，将这些日志推送到

Logstash 中进行处理，然后为其在 Elasticsearch 中创建索引。在我们搭建的环境中，Logstash 运行在 192.168.0.112 上，而 Kibana 运行在 192.168.0.111 上。这里的 Elasticsearch 实例与为实现 Liferay 搜索引擎所安装的 Elasticsearch 不同，它是低版本 1.4.0，因为这个版本是 Elasticray 插件所支持的版本。

使用 Metricbeat 和 Packetbeat 来采集数据，并且将数据直接送入 Elasticsearch。最终，使用 Kibana 可视化这些数据。

6.3.1 搭建 Elasticsearch

根据不同需求，可以将 Elasticsearch 设置为独立的节点或者集群。如果想搭建集群，可以参考第 11 章，对集群中的节点数量加以理解。

一旦在机器上获取 Elasticsearch 的软件包，并解压之后，就可以设置节点名称、日志存储路径等属性。修改配置文件 config/elasticsearch.yml 如下：

```
node.name: es-node-stack
path.data: /var/lib/elasticsearch
path.logs: /var/log/elasticsearch
network.host: 192.168.0.110
http.port: 9200
```

如果想创建集群，可以为 Elasticsearch 集群设置一个名称：

```
cluster.name: es-stack
```

配置 Elasticsearch 之后，即可运行这些节点。转到 Elasticsearch 解压的存储目录，并运行 Elasticsearch：

```
./bin/elasticsearch
```

> TIP 最佳的做法是将 Elasticsearch 作为系统服务安装和使用。要了解这方面的内容，请参阅第 1 章的安装部分。

上述操作将使 Elasticsearch 节点/集群启动并运行。Elasticsearch 现在已经准备好接收每个节点中 Beats/Logstash 发送的数据。

6.3.2 搭建 agents/Beats

可根据需要设置 Packetbeat、Metricbeat 和 Filebeat。如前面的表中所定义的内容，Beats 对应于更新后的体系结构中每个相应的节点，下面在每个服务器上安装各自的 Beats 组件。

6.3.2.1 Packetbeat

根据需要修改 Packetbeat 配置文件 Packetbeat.yml。由于要获取 HTTP 和 MySQL 协议的数据，因此需要为每个协议配置端口。我们需要选择一个网络接口来获取数据包，并从以太网获取数据，因此选择 en0，这也是接口设备的默认值，如下所示：

```
packetbeat.interfaces.device: en0
```

需要为 MySQL 和 HTTP 协议设置端口号：

```
packetbeat.protocols.http:
ports: [80, 8080, 8000, 5000, 8002]
packetbeat.protocols.mysql:
ports: [3306]
```

将这些数据直接传送到 Elasticsearch，Elasticsearch 集群 IP 是带有默认端口号的 192.168.0.110：

```
output.elasticsearch:
hosts: ["192.168.0.110:9200"]
template.name: "packetbeat"
template.path: "packetbeat.template.json"
template.overwrite: false
```

其中的 template(模板)用于在 Elasticsearch 中设置映像。可以将所有其他属性保持为默认值，并启动每台机器上的 Packetbeat。

6.3.2.2 Metricbeat

为了获取系统和服务器相关的统计数据，需要在网络拓扑中的每个服务器上配置 Metricbeat。可根据需要修改配置文件 metricbeat.yml。

对于每个服务器需要捕获系统信息和统计数据，metricbeat.yml 的配置如下：

```
metricbeat.modules:
-module: system
metricsets:
-cpu
-load
-core
-diskio
-filesystem
-fsstat
```

```
-memory
-network
-process
enabled: true
period: 10s
processes: ['.*']
```

对于 Nginx 服务器，也可以这样配置来检查其状态：

```
-module: nginx
metricsets: ["stubstatus"]
enabled: true
period: 10s
hosts: ["http://192.168.0.103"]
```

对于 MySQL 服务器，可以通过如下配置来检查其状态：

```
-module: mysql
   metricsets: ["status"]
   enabled: true
   period: 10s
   hosts: ["root@tcp(192.168.0.106:3306)/"]
```

对于主机配置，MySQL 数据源名称应该设置为以下格式：

```
[username[:password]@][protocol[(address)]]/
```

如果没有在数据源名称中指定用户名和密码，则应以如下所示的形式来指定：

```
username: <username>
password: <password>
```

最后，配置输出属性来指定 Elasticsearch 节点正确的地址：

```
output.elasticsearch:
   hosts: ["192.168.0.110:9200"]
```

6.3.2.3 Filebeat

通过设置 Filebeat 来读取来自所有服务器的日志。我们将会从不同的位置来读取文件，因此需要为安装的每个 Filebeat 设置不同的 prospector。

为所有的用例写入共同的配置：输出到位于 192.168.0.112 上的 Logstash。

```
output.logstash:
```

```
    hosts: ["192.168.0.112:5044"]
```

对于每一个用例,为安装于/usr/servers目录中的 Liferay 服务器设置 prospector(每个 prospector 以一个-(连字符)开头):

```
-input_type: log
  paths:
    -/usr/servers/liferay-portal-6.2-ce-ga4/logs/*.log
  document_type: lrlogs
```

我们也可以使用 Filebeat 标签来标记带自定义名称的事件。例如,如果希望区分来自 liferay 的日志,可使用标记名称 liferay-node-x,其中,x 是指节点的数量,例如 1、2,等等。

```
-input_type: log
  paths:
      -/usr/servers/liferay-portal-6.2-ce-ga4/logs/*.log
  tags: ["liferay-node-1"]
  document_type: lrlogs
```

在前面的配置中,将 Liferay 集群的第一个节点的标记设置为 liferay-node-1。对于另一个节点,可以设置其标签为 liferay-node-2。

因为 Liferay 是一个基于 Java 的服务器,并且在日志中也会含有 Java 堆栈跟踪,因此在这种情况下,需要加入多行日志配置。例如,下列日志条目显示 Liferay 的自动部署扫描器试图扫描 ZIP 文件,但由于 ZIP 文件损坏,或者由于权限问题无法打开文件,所以操作失败:

```
10:50:42,357 ERROR
[com.liferay.portal.kernel.deploy.auto.AutoDeployScanner][AutoDeployDir:220
] com.liferay.portal.kernel.deploy.auto.AutoDeployException:
java.util.zip.ZipException: error in opening zip file
com.liferay.portal.kernel.deploy.auto.AutoDeployException:
java.util.zip.ZipException: error in opening zip file
    at
com.liferay.portal.kernel.deploy.auto.BaseAutoDeployListener.isMatchingFile
(BaseAutoDeployListener.java:104)
    at
com.liferay.portal.kernel.deploy.auto.BaseAutoDeployListener.isHookPlugin
(BaseAutoDeployListener.java:46)
    ...
Caused by: java.util.zip.ZipException: error in opening zip file
    at java.util.zip.ZipFile.open(Native Method)
    at java.util.zip.ZipFile.<init>(ZipFile.java:215)
```

```
    at java.util.zip.ZipFile.<init>(ZipFile.java:145)
    at java.util.zip.ZipFile.<init>(ZipFile.java:159)
    at
com.liferay.portal.kernel.deploy.auto.BaseAutoDeployListener.isMatchingFile
(BaseAutoDeployListener.java:89)
    ... 6 more
```

对于这些类型的日志，要用一个模式来指定 Filebeat 配置，以匹配并将日志条目的所有后续行与主要的日志行合并。对于这些日志，以下是多行配置：

```
multiline.pattern: '^([0-9]{2}:[0-9]{2}:[0-9]{2},[0-9]{3})'
multiline.negate: true
multiline.match: after
```

现在，随着多行配置的加入，这类异常内容将会成为一个日志条目中的一部分。这个模式将在开始时匹配行，并且考虑一个新的日志条目，否则，不匹配的行将成为先前匹配的行的一部分。

除了这些日志条目之外，还有应用服务器生成的日志，这些日志有以下格式：

```
Nov 15, 2016 8:53:28 AM org.apache.catalina.startup
.HostConfig deleteRedeployResources
INFO: Undeploying context [/notification-portlet]
```

对于这些日志条目，多行配置如下：

```
multiline.pattern: '^([A-Z]{1}[a-z]{2} [0-9]{1,2}, [0-9]{4}
[0-9]{1,2}:[0-9]{2}:[0-9]{2} (AM|PM))'
multiline.negate: true
multiline.match: after
```

在这里，尝试用 MMM dd，yyyy HH：mm：ss 的形式匹配日期模式。为了便于理解，并在日志中提供灵活性，将在两个类型中存放这些日志——**lrlogs** 用于存放 Liferay 软件日志；**lrcatalinalogs** 用于存放应用服务器相关的日志。在 node-1 上的最终配置类似于：

```
-input_type: log
  paths:
    -/usr/servers/liferay-portal-6.2-ce-ga4/logs/*.log

  tags: ["liferay-node-1"]
  document_type: lrlogs
  multiline.pattern: '^([0-9]{2}:[0-9]{2}:[0-9]{2},[0-9]{3})'
  multiline.negate: true
  multiline.match: after
```

```
-input_type: log
  paths:
    -/usr/servers/liferay-portal-6.2-ce-ga4
    /tomcat-7. 0.42/logs/catalina*.log
  tags: ["liferay-node-1"]
  document_type: lrcatalinalogs
  multiline.pattern: '^([A-Z]{1}[a-z]{2} [0-9]{1,2}, [0-9]{4}
[0-9]{1,2}:[0-9]{2}:[0-9]{2} (AM|PM))'
  multiline.negate: true
  multiline.match: after
```

这两种不同日志类型的文件都位于不同的存储位置。与 Liferay 相关的日志在目录 LIFERAY_HOME/logs 中可以找到，与应用服务器相关的日志位于目录 tomcat-xx-xx/logs 中。在本例中，Liferay 可以从目录 LIFERAY_HOME 中提取，而 tomcat 当前所在的目录是/usr/servers/liferay-portal-6.2-ce-ga4。

Nginx 实例上的 Filebeat 配置较为简单，类似于如下配置：

```
-input_type: log
  paths:
    -/usr/local/var/log/nginx/access.log
  document_type: nginxaccesslogs
```

在 OpenDJ LDAP 服务器实例上的同种配置如下所示：

```
-input_type: log
  paths:
    -/usr/servers/opendj/logs/*
  document_type: ldaplogs
```

OpenDJ 日志条目本质上不是多行的，因此不需要多行过滤器。

对于 Elasticsearch 实例有：

```
-input_type: log
  paths:
    -/usr/servers/elasticsearch-liferay/logs/*.log
  document_type: elasticsearchlogs
```

6.3.3 搭建 Logstash

一旦 Filebeat 开始捕获日志，就可以运行 Logstash 来接收日志，解析、处理这些日志，并送入 Elasticsearch 创建索引。为什么需要 Logstash？ Filebeat 可以直接将日志发送到

Elasticsearch，因此它们看起来没有变化。

使用 Logstash 处理数据，可以更仔细地查看数据和已处理的字段，同时日志也可以提供更好的分析数据的机会。例如，如果分解 Nginx 日志来找出地理位置数据，那么在结果中会得到清晰的关于位置的信息，这些信息反映了大多数用户访问服务器时所在的地理位置。为了将日志分解成多个字段，可以使用 grok 模式。

在之前章节中，配置 Filebeat 时添加了这些类型的日志：nginxlogs、liferaylogs、elasticsearchlogs 和 ldaplogs，并使用 grok 处理 nginx、liferay 和 OpenDJ 日志。将在 logstash-5.1.1/conf 中创建名为 patterns 的目录存储所需的所有模式。

6.3.3.1 Nginxlogs 中的 grok

在 logstash-5.1.1/conf/patterns 中为此模式创建名称为 nginx.pattern 的文件。一个简单的访问日志的示例如下所示：

```
27.109.15.3 - - [02/Jan/2017:13:10:56 +0000] "GET /dashboard HTTP/1.1" 200
4755 "http://google.co.in/" "Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML,like Gecko) Chrome/55.0.2883.87 Safari/537.36"
```

这个日志条目是一个典型的 Nginx 访问日志，它有 IP、用户、认证、时间标记(timestamp)、请求方式、相对 URL 调用、HTTP 响应代码、内容长度、referrer 和 agent。对于这样的日志条目，我们的 grok 模式如下所示：

```
NGUSER [a-zA-Z\.\@\-\+_%]+
NGINXACCESSLOG %{IPORHOST:clientip} %{NGUSER:user} %{NGUSER:auth}
\[%{HTTPDATE:timestamp}\] "%{WORD:verb} %{URIPATHPARAM:request}
HTTP/%{NUMBER:httpversion}"%{NUMBER:response}(?:%{NUMBER:bytes}|-)
(?:"(?:%{URI:referrer}|-)"|%{QS:referrer}) %{QS:agent}
```

这样的配置将为 Nginx 添加一个名称为 NGINXACCESSLOG 的模式，可以在 Logstash 中使用这个模式。

6.3.3.2 Liferaylogs 中的 grok

我们搭建的 Liferay 运行在 Tomcat 上。与服务器相关的日志存储在 catalina 日志中。我们自定义的逻辑/liferay 相关的日志存储在 liferay 日志中。典型 liferay 日志如下：

```
08:15:29,024INFO[http-bio-8080-exec-1718][TimeEntryHelperImpl:267]
Getting Project Task List
```

这个日志包含关于时间、日志级别、线程、Java 类、行号和日志消息的信息。自定义 grok 模式如下所示：

```
TOMCAT_DATESTAMP %{HOUR}:?%{MINUTE}(?::?%{SECOND})
LOGTHREAD ([A-Za-z0-9_.-]+)
TOMCAT_LOG %{TOMCAT_DATESTAMP:timestamp} %{LOGLEVEL:level}\s*
\[%{LOGTHREAD:logThread}\]\[%{LOGTHREAD:class}:%{INT:line}\]\s*%{GREEDYDATA
:message}
```

此日志模式将读取 liferay 日志并将其存储在相应的字段中。另一个类型的日志是 catalina 日志，它主要与 Liferay 门户网站的 portlet 部署有关。一个典型的日志条目如下：

```
Nov 15, 2016 8:53:28 AM org.apache.catalina.startup.HostConfig
deleteRedeployResources
INFO: Undeploying context [/notification-portlet]
```

在先前的日志中，通知 portlet 正在被取消部署。为了读取这些日志，grok 模式可以如下所示：

```
CATALINA_DATESTAMP %{MONTH} %{MONTHDAY}, %{YEAR}
%{HOUR}:?%{MINUTE}(?::?%{SECOND}) (?:AM|PM)
JAVALOGMESSAGE (.*)
CATALINALOGMESSAGE %{CATALINA_DATESTAMP:datestamp}
%{DATA:class}%{LOGLEVEL:level}:\s* %{JAVALOGMESSAGE:message}
```

这个日志是多行的。Filebeat 和 Logstash 配置允许读取多行日志，并使用上述 grok 模式来正确地从日志行中分解字段。在下一节中，将看到完整的配置。

6.3.3.3 OpenDJ logs 中的 grok

以下是示例日志消息：

```
[29/Sep/2016:15:47:53 +0530] CONNECT conn=72 from=127.0.0.1:60395
to=127.0.0.1:1389 protocol=LDAP
[29/Sep/2016:15:47:53 +0530] BIND REQ conn=72 op=0 msgID=1 version=3
type=SIMPLE dn="cn=Directory Manager"
[29/Sep/2016:15:47:53 +0530] BIND RES conn=73 op=0 msgID=1 result=0
authDN="cn=Directory Manager,cn=Root DNs,cn=config" etime=0
[29/Sep/2016:15:47:53 +0530] SEARCH REQ conn=72 op=1 msgID=2
base="dc=rkg,dc=test" scope=sub filter="(objectClass=inetOrgPerson)"
attrs="uid"
[29/Sep/2016:15:47:53 +0530] SEARCH RES conn=73 op=1 msgID=2 result=0
nentries=1 etime=1
[29/Sep/2016:15:47:53 +0530] SEARCH RES conn=72 op=1 msgID=2 result=0
nentries=1000 unindexedetime=343
[29/Sep/2016:16:13:34 +0530] DISCONNECT conn=73 reason="Client Disconnect"
```

```
[20/Nov/2016:13:15:56 +0530] DISCONNECT conn=0 reason="Server Error"
msg="LDAP Request Handler 0 for connection handler Administration Connector
0.0.0.0 port 4444 was unable to register this client connection with the
selector: java.nio.channels.ClosedChannelException"
[28/Sep/2016:09:33:40 +0530] category=UTIL severity=NOTICE
msgID=org.opends.messages.runtime.21 msg=Installation Directory:
/usr/servers/opendj
```

OpenDJ 日志并不简单，它们在格式上也不共享，因此所有此类日志信息的 grok 都是不同的，只有开头的时间戳有相同的格式。有的日志用来记录连接、绑定信息，以及搜索一般的日志。以下是这类日志信息的完整的 grok：

```
TIMESTAMP_TZ
\[%{MONTHDAY}/%{MONTH}/%{YEAR}:%{HOUR}%{MINUTE}(?:%{SECOND})\s*%{ISO8601_TI
MEZONE}?\]

OPENDJACESSSEARCHRES %{OPENDJACESSUNBIND} result=%{NUMBER:result}
nentries=%{NUMBER:numberOfEntries}
%{RESULTTYPE}\s*etime=%{NUMBER:eventTime}

RESULTTYPE (%{WORD:resultType})?

OPENDJACESSSEARCHREQ %{OPENDJACESSUNBIND} base=%{GREEDYDATA:baseInfo}
scope=%{GREEDYDATA:scope} filter=%{GREEDYDATA:filterCondition}

OPENDJACESSBINDRES %{OPENDJACESSUNBIND} result=%{NUMBER:result}
authDN=%{GREEDYDATA:authDetails} etime=%{NUMBER:eventTime}

OPENDJACESSBINDREQ %{OPENDJACESSUNBIND} version=%{NUMBER:versionNumber}
type=%{WORD:bindType} dn=%{GREEDYDATA:directoryName}

OPENDJACESSUNBIND %{TIMESTAMP_TZ:timestamp} %{GREEDYDATA:requestType}
conn=%{NUMBER:connID} op=%{NUMBER:opID} msgID=%{NUMBER:messageID}

OPENDJACESSDISCONNECT %{TIMESTAMP_TZ:timestamp} %{WORD:requestType}
conn=%{NUMBER:connID}
reason="%{DATA:reason}"(\smsg="%{GREEDYDATA:messageInfo})?

OPENDJACESSCONNECT %{TIMESTAMP_TZ:timestamp} %{WORD:requestType}
conn=%{NUMBER:connID} from=%{IPORHOST:fromIP}:%{NUMBER:fromPort}
to=%{IPORHOST:toIP}:%{NUMBER:toPort} protocol=%{WORD:protocolName}
```

```
OPENDJLOGS %{TIMESTAMP_TZ:timestamp} category=%{WORD:categoryName}
severity=%{WORD:severityLevel} msgID=%{GREEDYDATA:messageID}
msg=%{GREEDYDATA:message}

OPENDJACESSUNBIND %{TIMESTAMP_TZ:timestamp} %{GREEDYDATA:requestType}
conn=%{NUMBER:connID} op=%{NUMBER:opID} msgID=%{NUMBER:messageID}
```

这些 grok 覆盖 openDJ 的所有日志消息格式。将这些模式保存在 logstash-5.1.1/conf/patterns/opendj.pattern 文件中。

6.3.3.4　配置文件

处理 liferay、Nginx 和 OpenDJ 节点日志的配置文件如下：

```
input {
    beats {
        port =>5044
        codec =>multiline {
            patterns_dir =>"./conf/patterns/"
            pattern =>"(^%{TOMCAT_DATESTAMP}|
                        ^%{CATALINA_DATESTAMP}|
                        ^%{TIMESTAMP_TZ})"
            negate =>true
            what =>"previous"
        }
    }
}
filter{
    if [type] =="lrlogs" {
        mutate {
            gsub =>['message', "\n", " "]
            gsub =>['message', "\t", " "]
        }
        grok {
            patterns_dir =>"./conf/patterns"
            match =>[ "message", "%{TOMCAT_LOG}"]
        }
    }
    if [type] =="lrcatalinalogs" {
        mutate {
            gsub =>['message', "\n", ""]
            gsub =>['message', "\t", " "]
```

```
        }
        grok {
            patterns_dir =>"./conf/patterns"
            match => [ "message", "%{CATALINALOGMESSAGE}"]
        }
    }
    if [type] =="nginxaccesslogs" {
        grok{
            patterns_dir =>"./conf/patterns"
            match =>{ "message" =>"%{NGINXACCESSLOG}" }
        }
        geoip {
            source =>"clientip"
        }
    }
    if [type] =="ldaplogs" {
        grok{
            patterns_dir =>"./conf/patterns"
            match =>{ "message" =>[ "%{OPENDJLOGS}",
                    "%{OPENDJACESSCONNECT}",
                    "%{OPENDJACESSDISCONNECT}",
                    "%{OPENDJACESSBINDREQ}",
                    "%{OPENDJACESSBINDRES}",
                    "%{OPENDJACESSSEARCHREQ}",
                    "%{OPENDJACESSSEARCHRES}",
                    "%{OPENDJACESSUNBIND}" ] }
            }
        }
    }
output {
    elasticsearch {
        hosts =>"192.168.0.110:9200"
        document_type =>"%{[@metadata][type]}"
    }
}
```

> TIP patterns_dir 的路径也可以添加为绝对路径。

我们来了解一下在这里要做些什么。在输入部分，正在监听 Filebeat。来自 Filebeat 的输入通过多行编解码器进行处理。对于多行日志消息，该处理过程自动为换行处添加\n 符

号，并在每一行 Java 堆栈跟踪信息的开头添加\t 作为占位符。对日志进行转换，将其中的\n 和\t 去掉，使所有的日志行在送入 grok 过滤器前被视为单行。过滤器处理后，日志将被分解为各个字段，然后存入 Elasticsearch。

对于 Nginx 日志，不需要在输入部分指定编解码器，并在过滤器中进行转换。同样，可以使用 NGINXACCESSLOG 模式来匹配日志。此外，将使用 geoip，这样就可以使用访问者的 IP 来获取其位置相关的数据。这样的配置将添加一个位置字段，它具有经度和纬度值。如果在输出部分没有指定任何索引名称，则索引名称将自动设置为 logstash-%{+YYYY.MM.dd}。Elasticsearch 插件中已经加载到 Logstash 的映像模板会自动将位置字段转换为 geo_point，这样其中的数据就可以用于类似瓦片地图的可视化内容中。

对于 OpenDJ 日志信息，提供了 grok 模式的数组，其中将选择第一个与信息匹配的 grok。

准备好配置文件后，带着配置文件的名称 portal.conf，执行以下命令启动 Logstash：

```
./bin/logstash -f conf/portal.conf
```

此时 Filebeat 开始将读取的日志信息发送到 Elasticsearch，这些信息可以通过 Kibana 查看。

6.3.4 设置 Kibana

这个例子中 Kibana 的 IP 地址是 192.168.0.111，Elasticsearch 的 IP 地址是 192.168.0.110。设置 Kibana 主机的名称为 kibana-node-stack。其配置文件 config/kibana.yml 的内容如下所示：

```
server.host: "192.168.0.111"
server.port: 5601
elasticsearch.url: "http://192.168.0.110:9200"
server.name: "Kibana-node-stack"
```

一旦 Kibana 设置并配置完成，就可以运行 Kibana 来可视化输入的数据。在 Kibana 的安装目录中执行以下命令：

```
$ ./bin/kibana
```

Kibana 运行后，即可设置索引模式(index patterns)。例如，对于与日志相关的模式，所有的索引都将通过 logstash * 模式来匹配。

6.4 设置 Kibana 面板

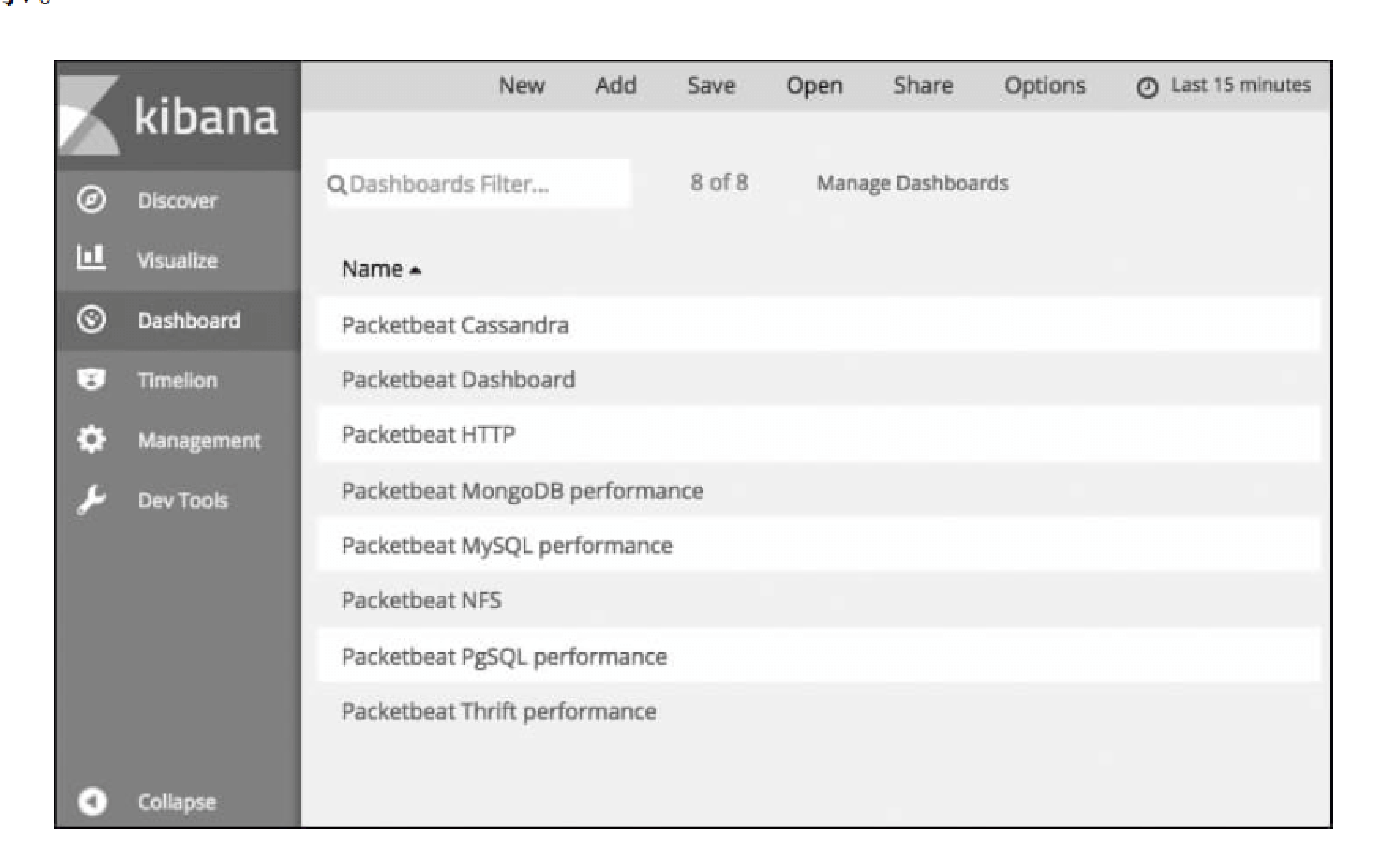
对于每一种 Beats 组件而言，官方提供了少量已创建的面板样本。可以使用各个 Beats 组件提供的脚本来导入这些面板。下面让我们来看一下每个 Beats 组件的面板。

6.4.1 Packetbeat

在 Packetbeat 存储目录中运行以下命令来导入 Packetbeat 的面板[①]：

```
./scripts/import_dashboards -es http://192.168.0.110:9200
```

执行这一命令，将在 Kibana 中导入所有 Packetbeat 的面板。可以转到 Kibana 中的 **Web Interface** | **Dashboard** | **Open** 菜单下查看这些面板，并且可以看到一个预设面板列表，如下图所示。



默认情况下，所有的面板都会被导入。如果不想保存面板，可以使用**管理面板(Manage Dashboards)**链接来删除面板。

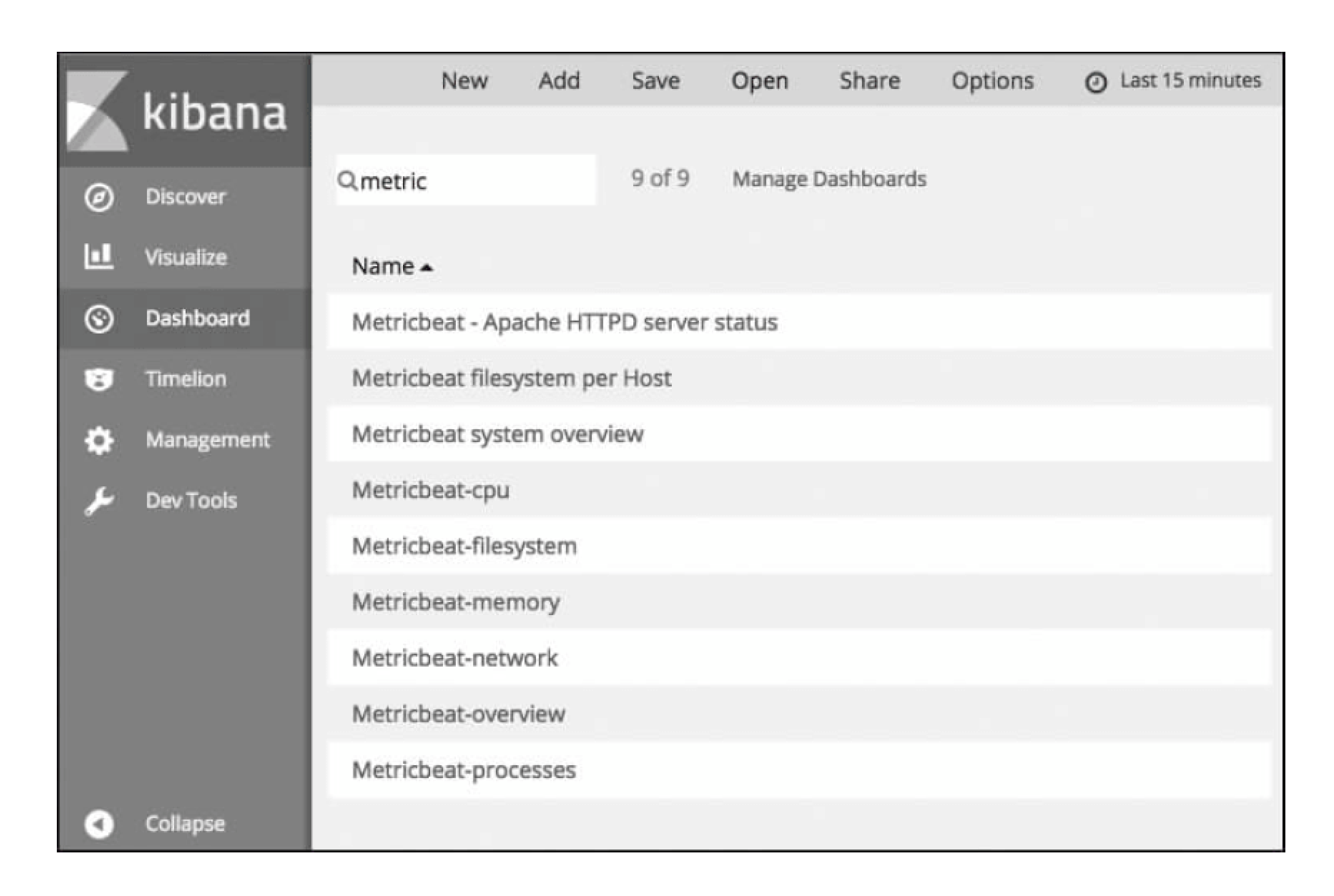
6.4.2 Metricbeat

在 Metricbeat 存储目录中执行以下命令导入 Metricbeat 的面板：

```
./scripts/import_dashboards -es http://192.168.0.110:9200
```

执行这一命令，将在 Kibana 中导入 Metricbeat 的所有面板。我们可以转到 Kibana 中的 **Web Interface** | **Dashboard** | **Open** 菜单下查看这些面板，并且你可以看到一个预设面板列表：

① 译者注：原文中这句话中间插入了一个＜indexentry dbid＝"92615" content＝"Packetbeat：importing，for Kibana Dashboards"＞标记，笔者认为这个标记与正文内容无关，故将其忽略。



默认情况下，所有的面板都会被导入。如果不想保存面板，可以通过使用**管理面板**链接来删除面板。

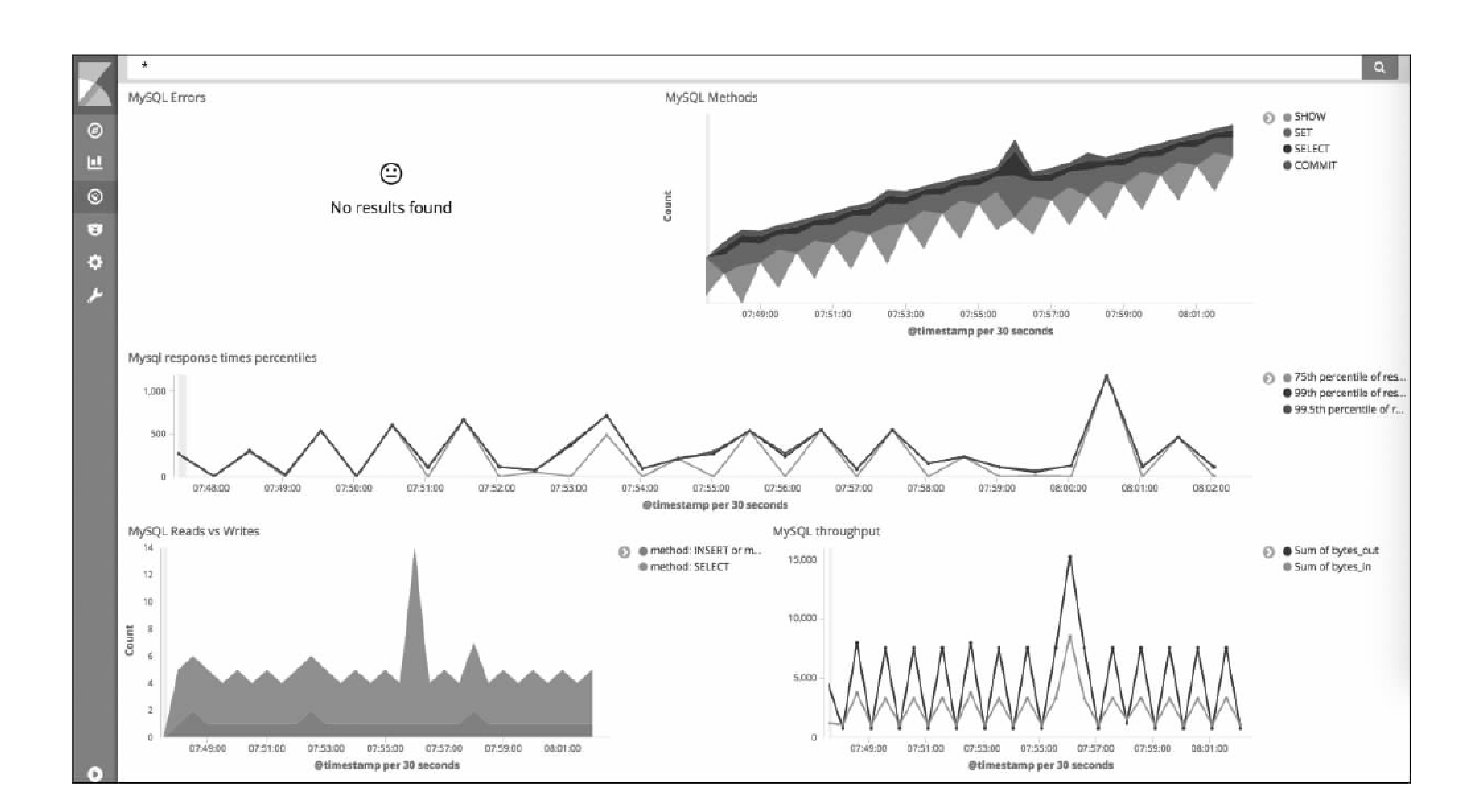
Filebeat 没有默认的面板，需要为获取的数据创建一个面板。

所有的组件全部搭建和运行并且所有的数据都已经获取之后，就已经做好了可视化的准备，并可捕获任何可能发生的问题。监视服务器的过程中，发生严重错误时很想了解每台机器的内存和处理器统计数据。对于数据库的统计数据，十分希望了解是否有些执行频繁但速度很慢的查询，以及响应时间、吞吐量等。幸运的是，只需花费很少精力，甚至不需要花费精力，即可对大部分统计数据执行可视化。导入面板时，所有的可视化内容都将被预装入Kibana 的面板中。

6.4.3 查看数据库(MySQL)性能

对于 MySQL，可以使用计数、响应时间、错误数量、读写对比、吞吐量等方法，如下图所示。这个面板可以转到 **Kibana** | **Dashboard** | **Open** | **Packetbeat MySQL Performance** 菜单下查看。

显示以上信息时，同一面板中也列出了最常见的查询和执行速度慢的 MySQL 查询，这通常有助于我们提高性能。可缓存常用的查询结果，并减少 DB 处理计数。对于执行速度慢的查询，可优化查询来改善响应时间。下面的截图中展示了两个列表：第一个列表显示了最频繁的查询，第二个列表按平均响应时间显示了所有查询中执行速度最慢的查询。



Most frequent MySQL queries

query: Descending	Count
SHOW FULL TABLES FROM `lportal_hrms` LIKE 'PROBABLYNOT'	300
SET autocommit=0	79
SET autocommit=1	79
commit	79
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED	32
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ	32
SET SQL_SELECT_LIMIT=1	32
SET SQL_SELECT_LIMIT=DEFAULT	32
select @@session.tx_read_only	32
SELECT AnnouncementsEntry.entryId as entryId378_0_, AnnouncementsEntry.uuid_ as uuid2_378_0_, AnnouncementsEntry.companyId as companyId378_0_, AnnouncementsEntry.userId as userId378_0_, AnnouncementsEntry.userName as userName378_0_, AnnouncementsEntry.createDate as createDate378_0_, AnnouncementsEntry.modifiedDate as modified7_378_0_, AnnouncementsEntry.classNameId as classNam8_378_0_, AnnouncementsEntry.classPK as classPK378_0_, AnnouncementsEntry.title as title378_0_, AnnouncementsEntry.content as content378_0_, AnnouncementsEntry.url as url378_0_, AnnouncementsEntry.type_ as type13_378_0_, AnnouncementsEntry.displayDate as display14_378_0_, AnnouncementsEntry.expirationDate as expirat15_378_0_, AnnouncementsEntry.priority as priority378_0_, AnnouncementsEntry.alert as alert378_0_ FROM AnnouncementsEntry WHERE (displayDate >= '2016-11-08 02:11:17.795') AND (displayDate <= '2016-11-08 02:26:17.793')	1

Slowest MySQL queries

query: Descending	Average responsetime
SHOW FULL TABLES FROM `lportal_hrms` LIKE 'PROBABLYNOT'	259.03
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED	82.188
SET autocommit=0	38.063
SELECT TRIGGER_NAME, TRIGGER_GROUP, NEXT_FIRE_TIME, PRIORITY FROM QUARTZ_TRIGGERS WHERE SCHED_NAME = 'PersistedQuartzSchedulerEngineInstance' AND TRIGGER_STATE = 'WAITING' AND NEXT_FIRE_TIME <= 1478571566204 AND (MISFIRE_INSTR = -1 OR (MISFIRE_INSTR != -1 AND NEXT_FIRE_TIME >= 1478571476204)) ORDER BY NEXT_FIRE_TIME ASC, PRIORITY DESC	21
SELECT COUNT(TRIGGER_NAME) FROM QUARTZ_TRIGGERS WHERE SCHED_NAME = 'PersistedQuartzSchedulerEngineInstance' AND NOT (MISFIRE_INSTR = -1) AND NEXT_FIRE_TIME < 1478572222461 AND TRIGGER_STATE = 'WAITING'	19
SELECT TRIGGER_NAME, TRIGGER_GROUP, NEXT_FIRE_TIME, PRIORITY FROM QUARTZ_TRIGGERS WHERE SCHED_NAME = 'PersistedQuartzSchedulerEngineInstance' AND TRIGGER_STATE = 'WAITING' AND NEXT_FIRE_TIME <= 1478571515105 AND (MISFIRE_INSTR = -1 OR (MISFIRE_INSTR != -1 AND NEXT_FIRE_TIME >= 1478571425105)) ORDER BY NEXT_FIRE_TIME ASC, PRIORITY DESC	7
SELECT TRIGGER_NAME, TRIGGER_GROUP, NEXT_FIRE_TIME, PRIORITY FROM QUARTZ_TRIGGERS WHERE SCHED_NAME = 'PersistedQuartzSchedulerEngineInstance' AND TRIGGER_STATE = 'WAITING' AND NEXT_FIRE_TIME <= 1478571700642 AND (MISFIRE_INSTR = -1 OR (MISFIRE_INSTR != -1 AND NEXT_FIRE_TIME >= 1478571610642)) ORDER BY NEXT_FIRE_TIME ASC, PRIORITY DESC	7
SELECT TRIGGER_NAME, TRIGGER_GROUP, NEXT_FIRE_TIME, PRIORITY FROM	7

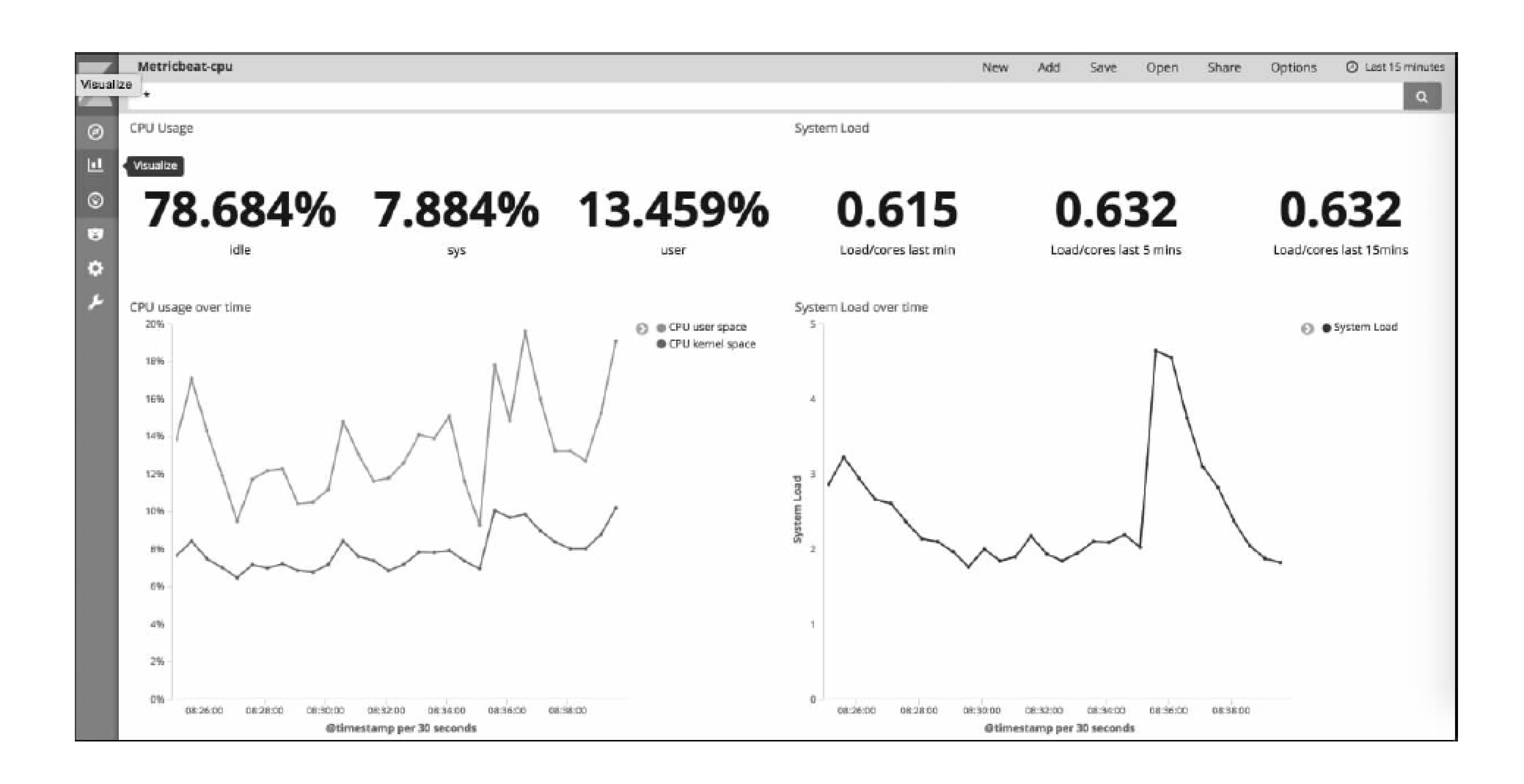
除了这些统计图表和数据列表，Packetbeat 中还提供了总览面板（Overview Dashboard），它还列出了 Web 事务、延迟、错误与成功事务的对比等。

若需要添加更多的可视化内容，可以通过新建可视化并添加至面板来完成。同样，也可以从面板中删除不必要或者不相关的可视化内容。

6.4.4 分析 CPU 的使用

使用 Metricbeat 面板，可以从 Metricbeat-CPU 面板看到 CPU 的使用率和系统负载，如下图所示。

除此之外，同一面板中还列出了搭建和运行 Metricbeat 的每台机器的 CPU 统计数据。



6.4.5　内存使用情况

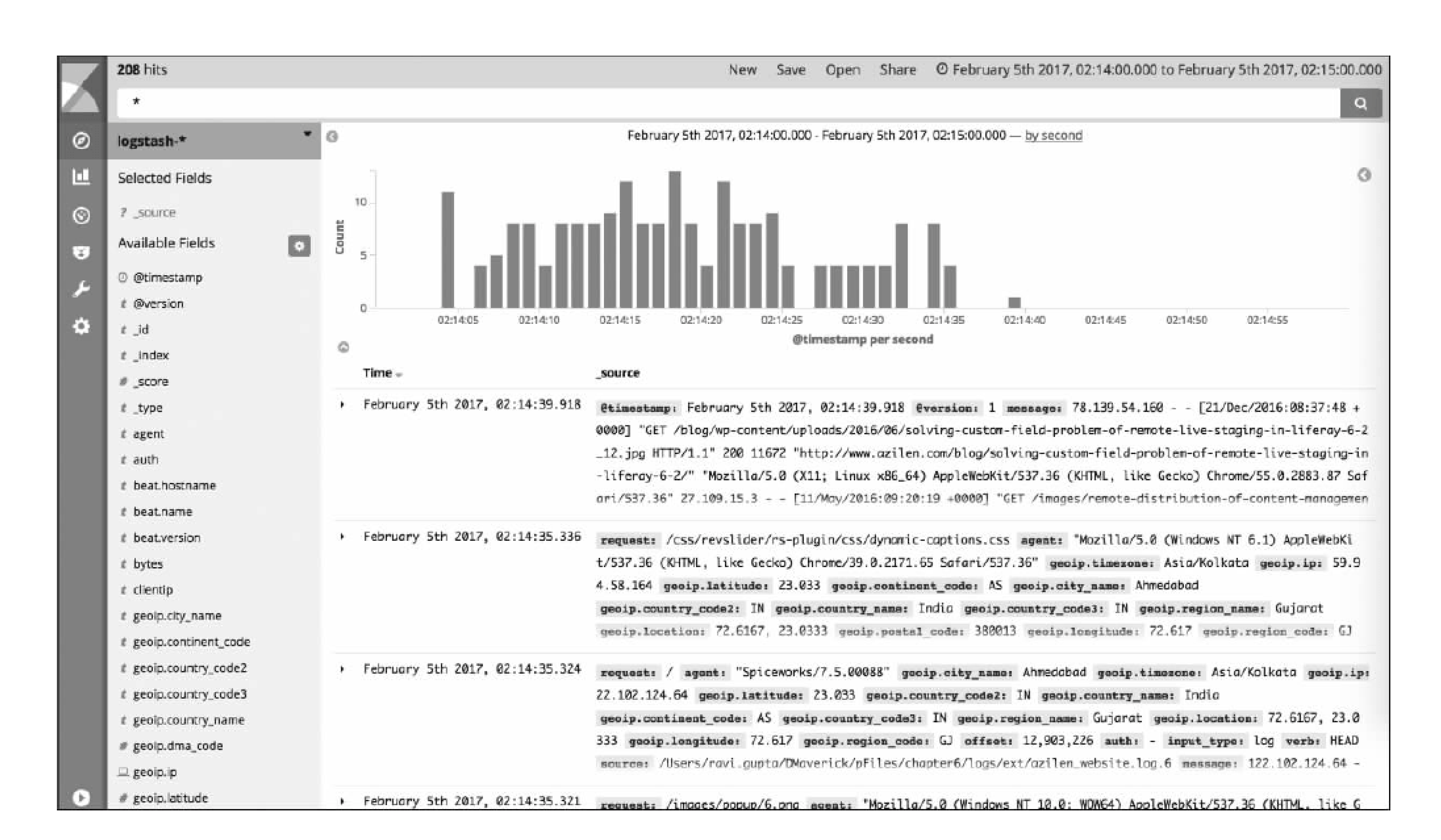
类似于 CPU 统计数据，Metricbeat 也提供了一个与内存相关的统计数据的面板。这个面板展示了内存占用量、可用内存容量、交换空间使用情况以及内存占用量最大的前几台主机的数据，如下图所示。



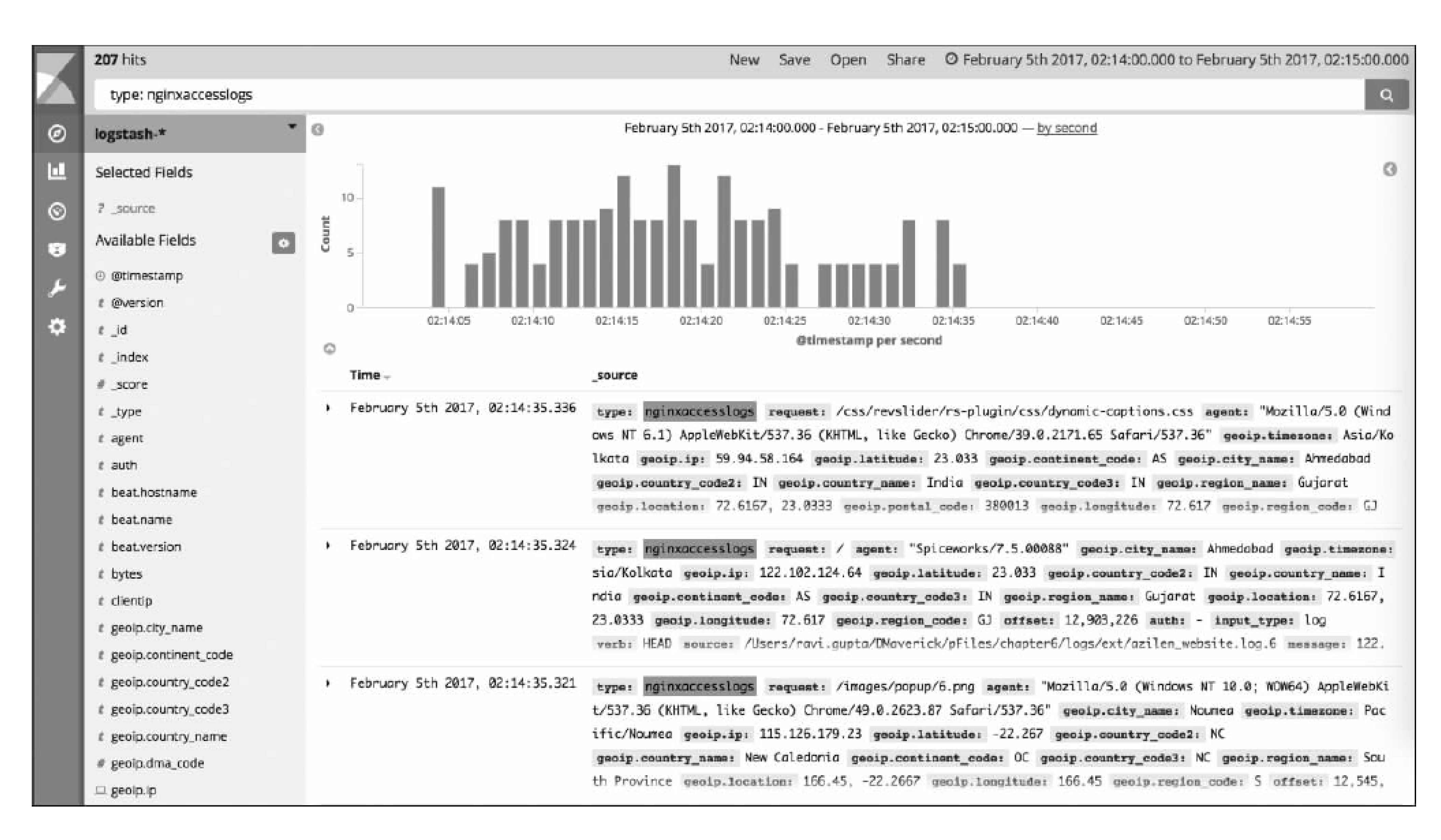
还有其他的可用面板，帮我们可视化与进程相关的数据。可以按内存和 CPU 使用情况来查看占用资源最多的进程，以及其中每一个进程的 CPU 和内存使用情况列表。

6.4.6　检查日志

当 Filebeat 正在运行时，它将搜集所有发生变化的日志文件内容，通过 Logstash 解析后发送到 Elasticsearch 中创建索引。转到 **Kibana** | **Discover**，可以看到所有匹配 **logstash** 的索引模式的日志。下图是一个日志列表的示例。



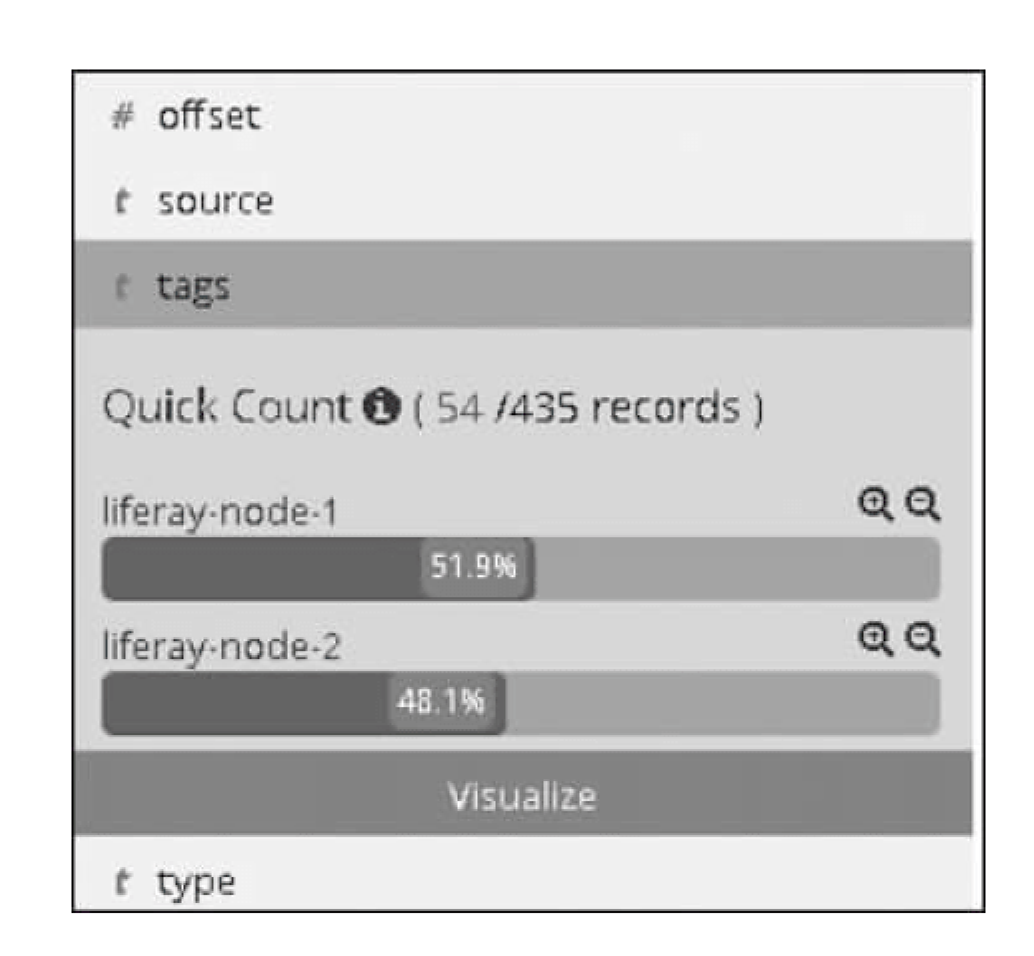
我们可以使用不同的参数(字段,在左侧导航栏中**可用字段 Available Fields** 下方列出)。通常情况下,在生产环境中,当有大量并发用户时,就会产生非常多的日志,甚至每秒或者每分钟产生上千条日志。在日志中发现一个错误是不容易的,这时,像 Kibana 这样的工具提供了一种功能,以使我们可以在其中搜索日志条目。从大量日志中搜集有用的信息是一项很艰巨的任务,分类是很有用的——就像设置 Filebeat prospectors 时使用文档类型一样。另外,也可为主要的应用服务器节点添加标签。下图是使用 **nginxaccesslogs** 类型过滤后的日志。



可使用 nginxaccesslogs 过滤日志。正如所看到的，只有 **nginxaccesslogs** 类型的日志被展示出来了。这个视图提供了一个特定的、定向的视图和要查找的日志列表。

类似地，也可以尝试标签。可使用 Filebeat 配置中指定的标记为 **liferay node 1** 选择日志。

对于给定的时间范围，还可以通过单击左侧的可用字段来查看日志的分类，可以看到每个类别的日志数量。例如，如果想知道 liferay 每个节点的日志数量，就可以在**标签**下看到，如下面的截图所示。



在这里可以很快地创建可视化。如果单击日志数量下面的可视化按钮，界面中将会出现一个直方图，liferay 节点在 X 轴上，计数在 Y 轴上。

类似于这些统计数据和日志，还可以注意所有的节点和服务器，包括应用程序服务器(我们案例中的门户网站)、HTTP 服务器、LDAP 服务器、搜索引擎等。

6.4.7 寻找访问最多的网页

如果服务于大量用户，就会希望访问最多的页面，以便能提供更好的内容和用户体验。Packetbeat HTTP 面板列出了所有网页的 URL 及其统计数据，如下图所示。

在该图中，可以看到一个 URL 列表及其请求类型和计数，从而可以清晰地看到，人们更感兴趣的是获取项目信息和访问它们的时间表。根据这些数据可以得出许多结论，这完全取决于所在组织的需要。

6.4.8 访客地图

了解访客来自哪里是一件令人好奇的事情，如果能展示在地图上，就更好了。谷歌 Web 分析的平台展示了这种可视化内容，可帮我们了解一个区域的访客数量。在 Kibana 和瓦片地图的帮助下，可以建立一个相似的可视化内容，其中包含了位置信息(即经纬度)。

Top 10 HTTP requests

query: Descending	Count
POST /group/mint/time-sheet	8,027
GET /group/mint/project-dashboard	6,013
GET /group/mint/projects	6,013
GET /	4,024
GET /group/mint/employee-listing	4,022
POST /poller/receive	2,068
GET /mint-theme/images/portlet/draggable_borderless.png	2,017
GET /group/mint/dashboard	2,013
GET /c	2,012
POST /web/guest/home	2,012

Export: Raw Formatted

在设置 Logstash 时，添加了一个 grok 模式来分解日志；并使用 mutate 添加两个或者更多字段来构建 geo_point 字段。使用这些，就可以在地图上绘制地点。让我们来为这个案例创建一个可视化的效果：

(1) 转到 **Kibana** | **Visualize** | **New** | **Tile Map**，选择 **logstash** 索引模式。

(2) 在 metrics 中，设置聚合方式为 Count。

(3) 选择唯一可用的 bucket 子聚合——**geo Coordinates**。

(4) 从字段中选择 **geoip. location**。

(5) 选择 Options 选项卡，将地图类型改为热力图并且单击 Run 按钮，即可看到包含所有访客的热力图。

(6) 将地图类型更改为缩放的圆形标记，并放大到某个区域或国家。

可以看到访客遍布整个地图。这种可视化内容可帮助我们更好地了解访客的地理位置，并更好地为他们服务。

6.4.9 一定时间范围内的访客数量

查看网站的实时访问量是一件既有趣又重要的事情。可以设置一个指标来找出给定时间范围内的独立访客数量，这类指标有助于了解在什么时间范围内访客最多或最少。如果需要将网站关闭一段时间进行维护或停机，就可以看到访客量最低的时间统计：

(1) 选择所需的时间范围。

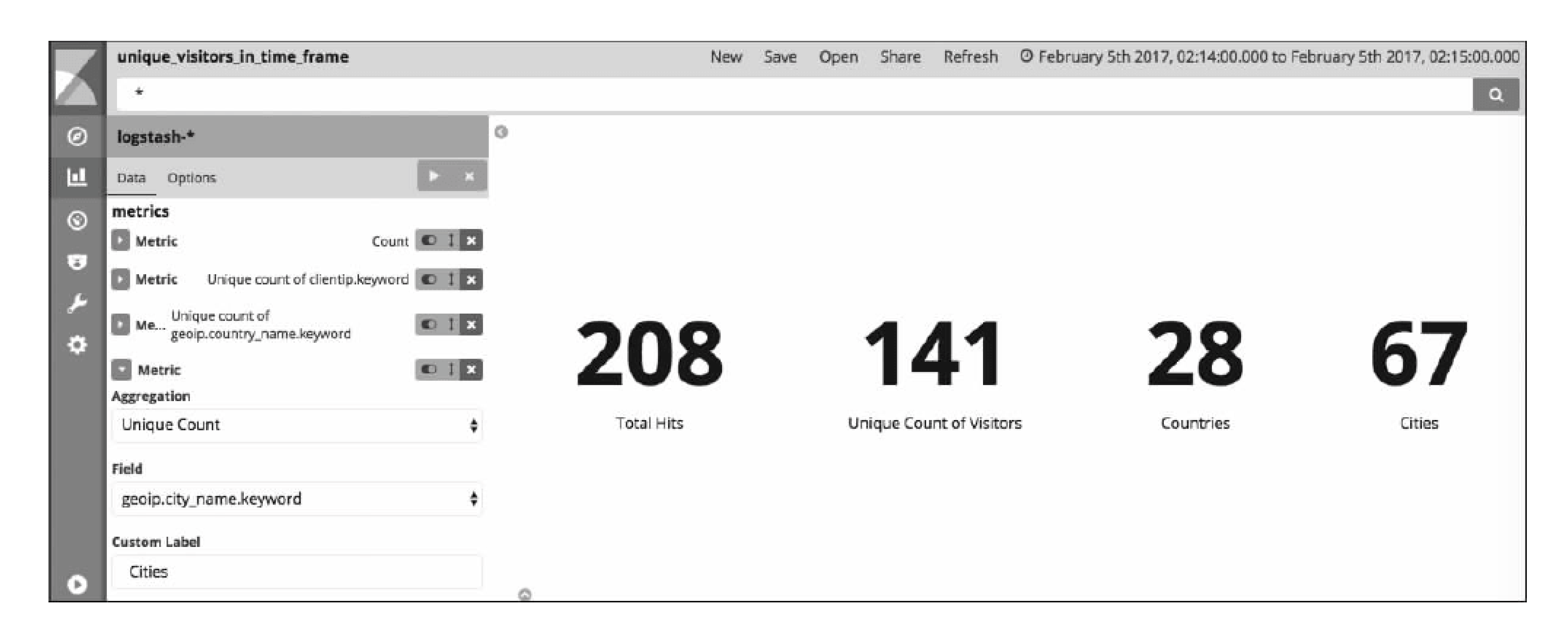
(2) 转到 **Kibana** | **Visualize** | **New** | **Metric**，选择 **logstash** 索引模式。

(3) 在 Metric 中，设置聚合方式为 Count，设置自定义标签为 **Total Hits(总命中量)**。

(4) 添加另一个 Metric，设置聚合方式为 **Unique Count**。选择 **clientip. keyword** 字段，并将标签设置为 **Unique Count of Visitors**。

(5) 类似地,可以在 **geoip.country_name.keyword** 字段上将更多的 metrics 聚合方式设置为 Unique Count。对于独立的城市计数,可以设置在 **geoip.city_name.keyword** 字段上。

(6) 单击运行按钮,会看到指定时间范围内的各项统计数据,如下图所示。



正如所看到的,来自 **28** 个国家 **67** 个城市的 **141** 位独立的访客执行了 **208** 次单击。此时,可以得到这样的数据,并查看网站上的流量。

6.4.10 请求类型

下面为所有请求方式的类型(如 GET、PUT、POST 等)实现另一种可视化。这是又一个需要了解的重要内容。大多数情况下,执行一个操作时,POST 请求就会被触发。这种数据有助于理解访客行为:

(1) 转到 **Kibana** | **Visualize** | **Data Table**,选择 **logstash** 索引模式。

(2) 在 metric 中,设置聚合方式为 Count。

(3) 添加一个 Split Row 类型的 bucket 聚合,并设置其方式为 Terms。

(4) 选择 verb.keyword 字段,并将标签更改为 **Request Type**(**请求类型**)。

(5) 单击运行按钮,界面中将生成下面的数据表:

Request Type	Count
GET	3,290
POST	202
HEAD	66

正如所看到的,相对于 POST 请求,大多数请求都是为了获取数据。

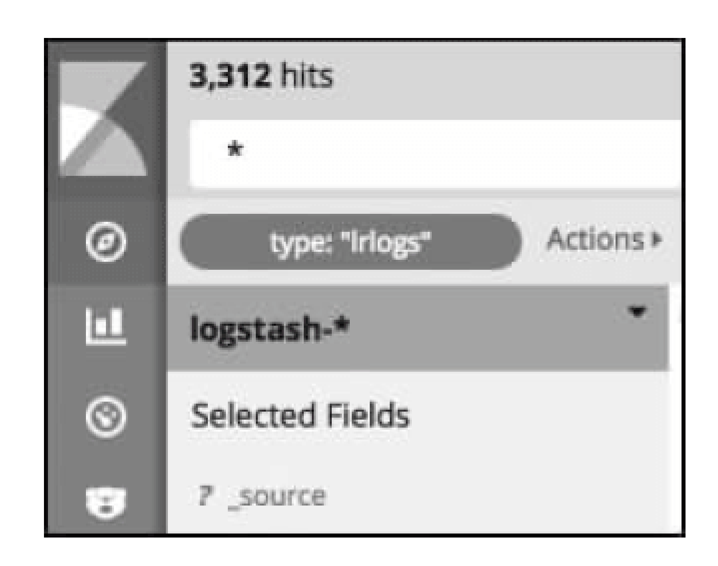
6.4.11 错误类型——日志的级别

类似于请求类型,也可以为日志级别设置一个数据表。对于这种可视化,可以为 lrlogs

设置一个过滤器，然后对 level.keyword 字段创建一个类似于请求类型那样的数据表。由于只关心为应用服务器节点(liferay 节点)所生成日志的级别，所以应该为 lrlogs 创建一个过滤器。要创建过滤器，可以按照以下步骤操作：

(1) 转到 **Kibana** | **Discover**，单击可用字段下的 type。

(2) 在这些类型中，单击 lrlogs 的带有＋号的放大镜图标，这一操作将添加一个类型为 lrlogs 的过滤器：



这个过滤器的定义如下：

```
{
    "query": {
        "match": {
            "type": {
                "query": "lrlogs",
                "type": "phrase"
            }
        }
    }
}
```

(3) 将这个过滤器固定，这样过滤功能就会在 Visualize 模式下被设置为可用。将会用它来可视化错误类型。



上图中第一个图像是添加过滤器时默认的设置，第二个图像是将鼠标悬停在其上方时的样子，最后一个是单击图钉的图标来固定这个过滤器时的样子。

设置过滤器之后，可以对 Liferay 日志中的级别实现可视化。创建可视化内容的步骤如下：

(1) 转到 **Kibana** | **Visualize** | **Data table**，选择 **logstash** 索引模式。

(2) 在 metric 中,设置聚合方式为 Count。

(3) 添加一个 splitrow 类型的 bucket 聚合,设置其方式为 Terms,应用在 level.keyword 字段上。

(4) 将标签修改为 **Log Level**,大小设置为 **10**,单击运行按钮。程序将按照日志级别及其出现次数的统计数据来生成数据表。

Log Level	Count
INFO	127
SEVERE	39
WARNING	2

可以根据需要更改时间范围。仅了解日志级别是不够的,还需要做更多的工作,例如,无论何时发生错误,都会得到通知。我们将在本章后续的内容中使用 Logstash 的电子邮件功能。

6.4.12 首选的 referrer

了解网站有哪些 referrer,理解和利用这些信息,对于搜索引擎优化是非常重要的。从可信赖且受欢迎的网站上链接到自己的站点是一件大事,你可能需要更多这样的引用。下面找出首选的 referrer:

(1) 移除已存在的过滤器(如果有的话)。

(2) 转到 **Kibana**|**Visualize**|**Data table**,选择 **logstash** 索引模式。

(3) 类似于其他数据表,仅修改 Terms 聚合的字段为 referrer.keyword。

(4) 将大小改为 **10**,单击运行按钮,将得到前 **10** 个 referrer 及其统计数据。

6.4.13 首选的代理 agent

找出首选的访问客户端代理,可以了解为访客提供最好的用户体验时需做的更改工作。类似于首选的 referrer 数据表,保持 agent.keyword 字段上的 Terms 聚合不变,为访问客户端创建一个可视化数据表。创建的可视化内容应如下图所示。

Agent Name	Count
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36"	121
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36"	85
"Mozilla/5.0+(compatible; UptimeRobot/2.0; http://www.uptimerobot.com/)"	78
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36"	53
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.112 Safari/537.36"	48
"Mozilla/5.0 (Windows NT 6.1; rv:40.0) Gecko/20100101 Firefox/40.0"	46
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36"	43
"Pingdom.com_bot_version_1.4_(http://www.pingdom.com)"	41
"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36"	36

可以看到，大多数请求都是在 Mac OSX 系统的客户端完成的。

6.5 使用 Logstash 电子邮件功能发警报

Alerting 是分析的一个关键部分。由于不可能一周 7 天 24 小时持续监控日志或任何其他统计数据，所以需要在某些特定事件发生时被提醒。例如，如果日志中有错误发生，就希望得到通知。没有人可以忍受生产环境服务器上的错误。

当某些错误发生时，可用 Logstash 的电子邮件功能发送邮件通知。为此，将使用 Logstash 的电子邮件输出插件。这个插件不是 Logstash 软件包的一部分，但是可以执行如下命令安装它：

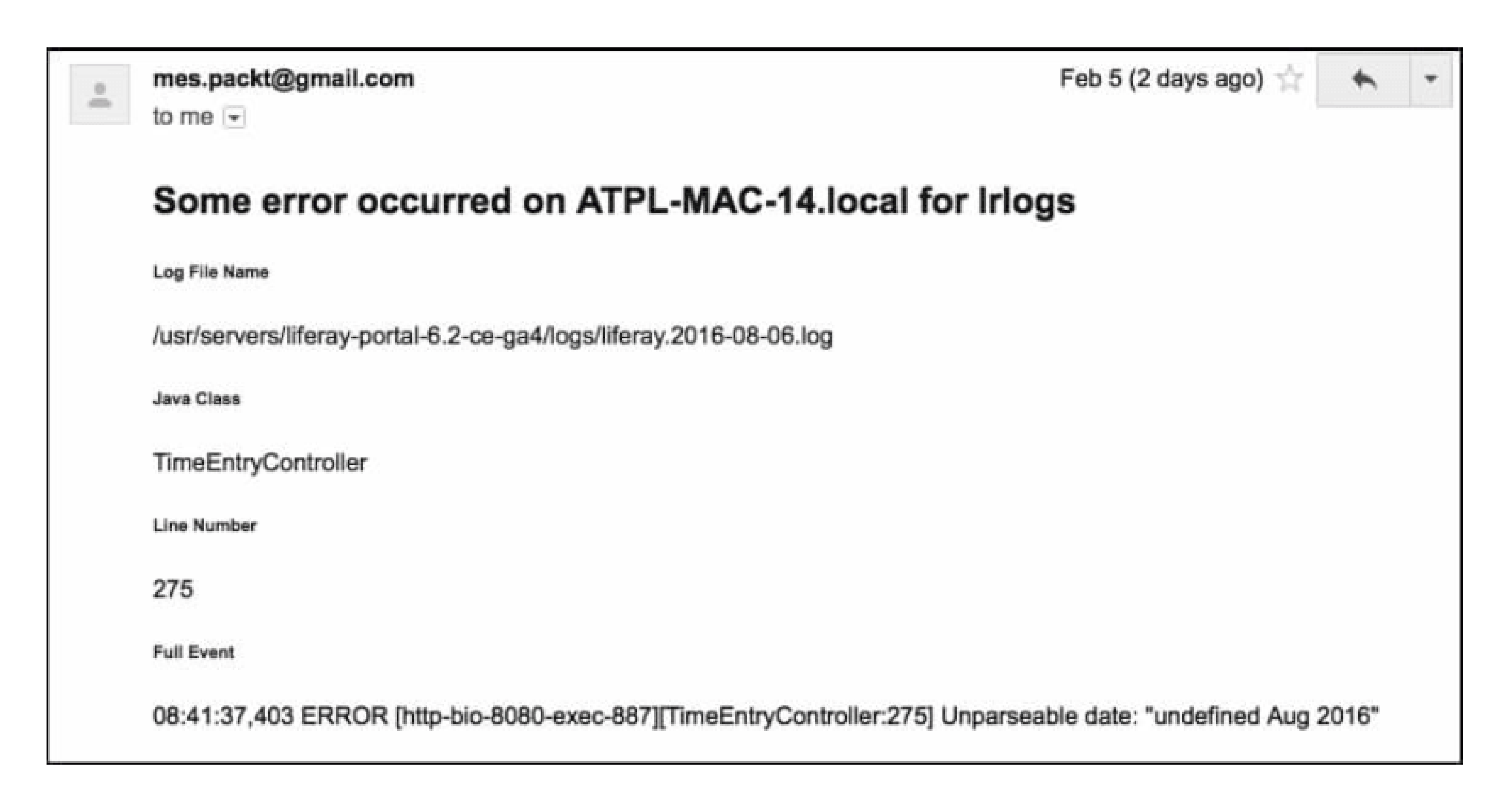
```
bin/logstash-plugin install logstash-output-email
```

一旦插件安装好，就可以在输出环节使用它。在配置文件中的输出部分，添加如下配置信息，配置 gmail 账户来发送邮件：

```
if [level] =="ERROR" {
    email {
        address =>"smtp.gmail.com"
        port =>"587"
        username =>"mes.packt"
        password =>"<password>"
        use_tls =>"true"
        from =>"<mes.packt@gmail.com>"
        subject =>"Error status"
        to =>"<to-email>"
        htmlbody =>"<h2>Some error occurred on %{host} for %{type}</h2>
                    <h6>Log File Name</h6>
                    <div>%{source}</div>
                    <h6>Java Class</h6>
                    <div>%{class}</div>
                    <h6>Line Number</h6>
                    <div>%{line}</div>
                    <h6>Full Event</h6><div>%{message}</div>"
    }
}
```

在上面的表单中，将检查日志级别是否发生了错误，这个字段通过 grok 过滤器添加。用确切的密码替代<password>，更改 to-e-mail 为实际的电子邮件地址。一旦错误发生，程序将会向指定的邮箱发送邮件。

例如，由错误日志触发的电子邮件功能见下图：



X-Pack 提供了使用 watcher 来发送通知的方法，将在第 10 章 X-Pack 插件中的 Alerting、Graph 和 Reporting 组件部分以及第 12 章案例分析——Meetup 中学习有关 watcher 的使用。

6.6 使用消息代理

实际服务器上的流量有时是很高的。当这种情况发生时，日志条目和统计数据就是非常重要的，这些数据的总量也很高。所有的 Beats 组件都将完成各自的工作，并将各自的数据发送到 Elasticsearch，但是有可能一些数据包/数据在处理过程中丢失了。这种情况可能是网络故障、数据高峰或其他原因造成的。重点是，索引的数据不能在任何情况下丢失。

要解决这个问题，使用消息代理或缓冲是一个不错的选择。适合的消息代理的工具有很多。对于开源工具来说，有两个很好的工具：

- **Redis**：http://redis.io/；
- **Kafka**：https://kafka.apache.org/。

有时消息代理似乎是生产环境所必需的，但在有些情况下，并不需要任何消息代理。使用 Filebeat 时，它充当临时缓冲区。生成的日志一定会被索引，但是可能需要一些时间，而且它的搜索可能会有一些延迟。如果这个延迟是可接受的，那么就没有必要为日志使用代理了。数据将保存到另一个硬件设备或服务器中，以监视生产环境中的运行情况。使用缓冲/消息代理纯粹依赖于系统需求。

6.7 本章小结

在本章中，我们分析了如何在生产环境中使用 Elastic Stack 组件。或多或少，在大多数生产环境中，都配置了类似的架构和安排。为服务器选择 Beats 组件时，应该特别谨慎。有时，可能会使用 Beats 来读取数据，但是这些数据并不是那么重要，并且对它们的管理完全可以采取其他手段。

在本章靠后的部分中，介绍了预定义面板如何帮助我们了解所需的信息。尽管这些面板提供了大部分有用的信息，但依然可以根据方便程度和需求自定义它们。

在下一章中，将学习如何自定义 Elastic Stack 中的每个组件，学会根据需要对它们进行个性化定制和修改。

第7章 个性化定制 Elastic Stack

判断软件工具是否功能强大的一个标准，就是看针对如下问题的解决方案：它是否能进行个性化配置？如果你正在推销软件产品，并且针对该问题的答案是“是的，我们就是这样做的。”那么，就可以较好地推销你的软件产品。这是一个开源软件产品的时代，一个软件产品的个性化定制水平如何，在决定它在适用用户的实际、个性化需求方面，起到了很大的作用，因为软件购买者可能会在很多方面（如自己定义个性化插件，修改已经存在的软件配置，修改用户界面等）需要通过个性化的设置，来自定义软件功能和外观。

在本章中，我们将会学习如何通过扩展或修改组件的方式，个性化定制 Elastic Stack。本章主要内容如下：

- 扩展 Elasticsearch；
- 扩展 Logstash；
- 扩展 Beats；
- 扩展 Kibana。

7.1 扩展 Elasticsearch

Elasticsearch 是一个拥有诸多优良特性的、稳定的、成熟的工程项目，而且它还是在不停的更新完善中。像其他优秀的开源软件一样，Elasticsearch 提供了扩充功能和允许用户以其个性化方式来配置 Elasticsearch 的途径。通过个性化设置，可以完成很多功能，这里列出一些。例如：通过个性化设置，使 Elasticsearch 能连接到一种新的数据仓库，或利用它来保存数据；可能会自己来写一个词法分析器 analyzer，和 Elasticsearch 已有的词法分析器相比，它能完成一些其他辅助功能；等等。在 Elasticsearch 的早期版本（V5 以前的版本）中，开发者往往需要生成自己的 rivers，以便能支持一种新的数据源。事实上，通过使用 HTTP 和 Transport 模块，可提供给软件开发人员更多的个性化设置的机会，使之能在标准 Elasticsearch 包代码之外，扩展 Elasticsearch 的功能，并能提供一些机制，使开发者能借助它，将其他的一些工具和技术集成到 Elasticsearch 开发环境中。

在 Elasticsearch 的开发者社区中没有太多的个性化定制软件的举措，下面列出的这些，是最常使用到的、比较流行的两种个性化 Elasticsearch 的方法：

- **写插件**：在 5.0 版本之前（当时软件名为 ELK Stack），它支持三种类型的插件：Java、Site 和 Mixed 的插件。Site 和 Mixed 类型的插件用于提供给用户一个 Elasticsearch 界面，并将 Web 开发技术与面向开发的语言结合起来。这些插件多数用于洞察数据索引 indices、数据分片 shards、数据的其他相关信息等。另一方面，Java 插件用于提供附加到 Elasticsearch 的特征，但它没有用户界面。这些研发出来的插件多用于支持在云环境中发现相关的聚类节点，对数据仓库、词法分析器等提供支持等。从 Elasticsearch 5.0 版本（此时软件新命名为 Elastic Stack）以后开始，Site 和 Mixed 类型的插件被弃用了，只留下了 Java 类型的插件。可以在如下地址找到所有 Elasticsearch（5.1.x 版本）中的所有可用插件：https://github.com/elastic/elasticsearch/tree/5.1/plugins。
- **利用 Elasticsearch 来扩展应用程序**：第二种方法是利用 Elasticsearch 提供的 API，来个性化定制 Elasticsearch。通过生成一个应用程序，使用 HTTP 或 Transport 模块调用 Elasticsearch 的 API，完成个性化定制。如果需要调用 HTTP 模块，可以通过使用简单的 REST 调用的方式来实现。另一方面，通过使用 Transport 协议，调用 Elasticsearch 的 API，可以在工程中增加项目依赖（dependency）。基于 Transport Client 的 JAR 文件可以在这里得到：https://mvnrepository.com/artifact/org.elasticsearch.client/transport/5.1.1。通过对 Site 和 Mixed 类型插件的部署（注：这些插件可使你能操纵 Elasticsearch 的数据索引 indices、数据分片 shards、聚类相关数据等），可以构建自己的客户端程序，并连接到 Elasticsearch 的集群中，完成标准的文件索引 index、搜索 search、数据获取 get 和删除 delete 等操作。

虽然未生成任何面向 Elasticsearch 的具体插件，但将在下面介绍如何构建开发环境，并介绍相关插件的结构。

7.1.1 Elasticsearch 开发环境

为了能通过生成插件来扩充 Elasticsearch 的使用功能，应该完成一些设定，以便加速研发的进程。第一件事就是按照如下方式得到 Elasticsearch 的源码：

```
$ git clone https://github.com/elastic/elasticsearch.git
```

上述命令将会检出（checkout）其中的主分支（master branch），其内容为正在研发的版本（通常不稳定）。使用如下命令，可以检出 Elasticsearch 5.1 版本。

```
$ git checkout -b elasticsearch-local origin/5.1
```

> 在这种情况下，不必从 https://git-scm.com 安装 Git。类似地，Gradle 可以从 https://gradle.org/得到。

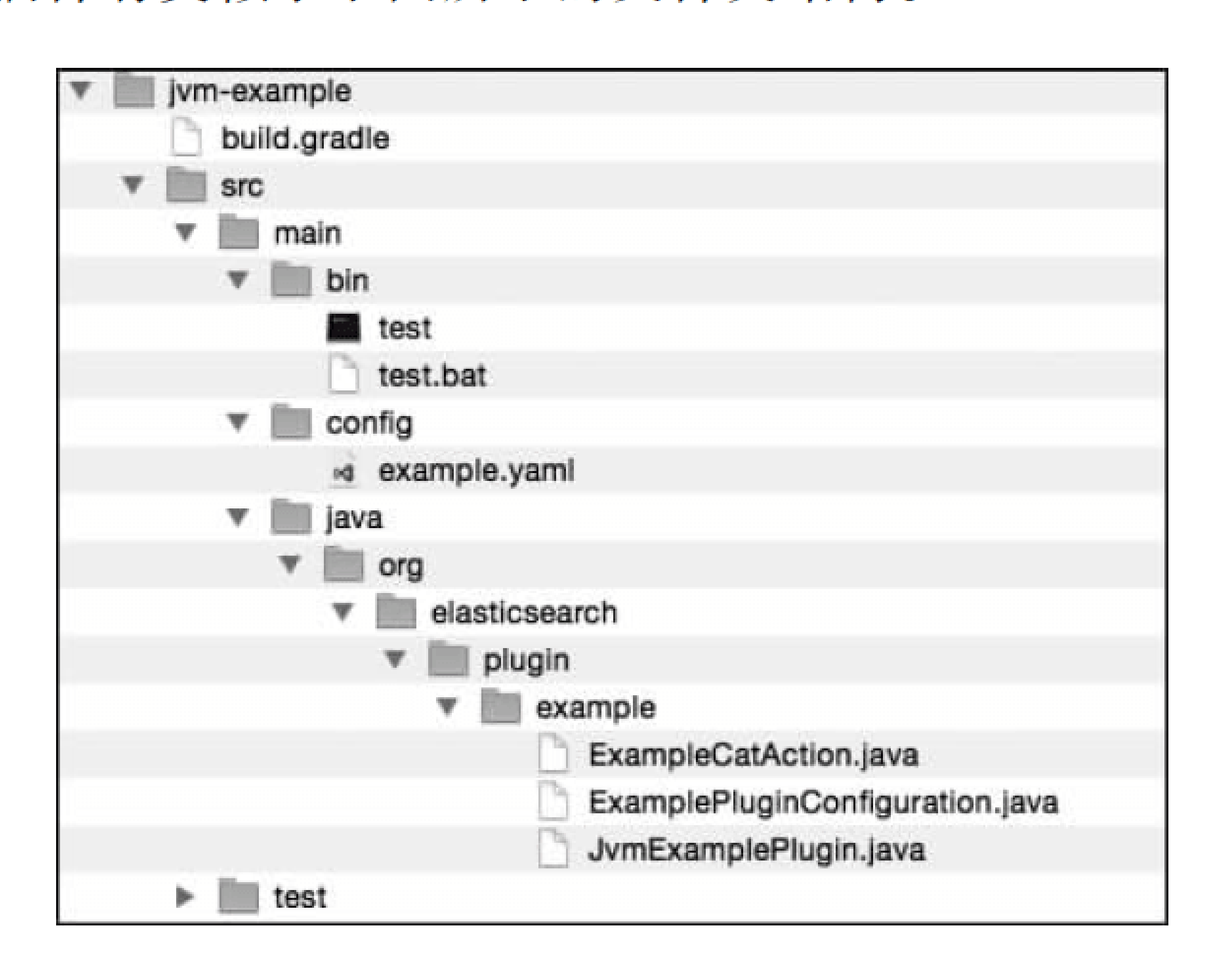
Elasticsearch 插件工程是一个基于 gradle 的工程[1]。为了开发和构建插件，我们需要 Java 和 Gradle。对于 Elasticsearch 而言，需要 Java version 8 以及 Gradle version 2.13 版本。你可以选用某个支持 Gradle 的 IDE 开发环境来完成对插件的开发，良好的 Elasticsearch 应用开发是很有必要的。

7.1.2　剖析一个 Elasticsearch Java 插件

Elastic 开发者团队已经有意识地在 Elasticsearch 的源码中给出一个名为 jvm-plugin 的插件例子(可从如下地址下载)：

```
https://github.com/elastic/elasticsearch/tree/5.1/plugins/jvm-example
```

一般情况下，该插件有类似于下图所示的文件夹结构。



该插件通常是一个基于 Gradle 的工程，包含一个配置文件 configuration 和一个插件类 plugin class。如果要开发自己的插件，则新开发的插件的文件夹结构应该与其类似。在 src/main 文件夹下，应该有一个 config 文件夹，在这里应该有一个 YAML 格式[2]的配置文件及一个包括运行该插件的 Java 文件夹。这个配置文件应该包括所有的和该插件运行有

① 译者注：Gradle 是基于 Apache Ant 和 Apache Maven 概念的项目自动化构建工具。它使用一种基于 Groovy 的特定领域语言来声明项目设置，抛弃了基于 XML 的烦琐配置。

② 译者注：YAML 是以数据(而不是以置标语言)为中心的标记语言。

关的配置信息，其中的 bin 和 test 文件夹则分别对应于二进制的和测试的用例文件夹。

作为例子，这里的主要插件类是 JvmExamplePlugin，它扩展自 org. elasticsearch. plugins 插件类，其余新的插件类也应当如此。

Gradle 的插件 esplugin 用于构建插件。在 build. gradle 文件中的开头部分，是一些必要的用于实现插件的设置内容，类似如下这样：

```
esplugin {
    description 'Demonstrates all the pluggable Java entry points in
    elasticsearch'
    classname 'org.elasticsearch.plugin.example.JvmExamplePlugin'
}
```

这里的类名是指构建的插件类的名字。它应该和插件类中的包名以及类名相匹配。

每一个 Elasticsearch 插件都必须在相应的 src/main/resource 文件夹下包含 plugin_descriptor. properties 文件。可能需要生成这个文件夹和文件。这个文件的作用在于使 Elasticsearch 知道这是一个插件类型的工程。默认情况下，该描述文件中的部分属性是可用的（属性为 enabled），例如，插件描述 plugin description、插件版本 plugin version、名称 name、类名 class name、Elasticsearch 版本、Java 版本等。由于 Elastic Stack 中所有组件的版本是相同的，因此添加的相应插件的版本应该兼容于同样版本的 Elasticsearch。对每一个新增加的插件，这都是强制性的要求。

```
description=Demonstrates all the pluggable Java entry points
version=5.1.1
name=jvm-example
classname=org.elasticsearch.plugin.example.JvmExamplePlugin
java.version=1.8
elasticsearch.version=5.1.1
```

7.1.3 构建插件

一旦完成上述准备工作，就可以使用 gradle 的标准命令来构建自己的插件代码了。如果通过运行 gradle 命令来构建插件工程，插件描述文件会自动创建，并放置在相应的插件文件夹中。

要构建源代码，需要定位到相应的插件文件夹，并运行如下命令：

```
gradle clean build
```

首次运行上述命令后，系统会搜集相应工程的所有工程依赖（dependency），编译完整的源码，并构建插件工程。

要生成用来部署的JAR文件，可执行如下命令：

```
gradle jar
```

执行上述命令，会在相应插件的build文件夹中生成JAR文件，并会产生5种类似下面所示的文件：

```
jvm-example-5.1.1-SNAPSHOT-javadoc.jar
jvm-example-5.1.1-SNAPSHOT-sources.jar
jvm-example-5.1.1-SNAPSHOT.jar
jvm-example-5.1.1-SNAPSHOT.pom
jvm-example-5.1.1-SNAPSHOT.zip
```

ZIP文件是其中的主要文件，它包括插件的JAR文件、插件描述文件、插件配置文件等。和其他任何类型的插件一样，这个新生成的插件可安装到Elasticsearch中。

作为一个开源软件社区，让大家能共享所做的工作，是一件很好的事情。生成一个自己的插件后，将该插件的代码放到某个宿主计算机(如可能会是github.com)上，以便社区成员都能充分利用该软件，这是一件非常不错的事情。

7.2 扩展 Logstash

对于Logstash，有两种扩展其功能的方法：一种是修改其内核，另一种是在其数据仓库中增加一个或多个自定义的插件。前文已经提到，Logstash共有4种类型的插件：输入Input、输出Output、过滤Filter、转换Codec。其实，在Logstash中自定义生成一个插件也是非常容易的(仅需要运行一个命令)；在新生成插件时，只需要提供它执行的具体业务逻辑。

Logstash插件是用Ruby程序开发语言(http://www.ruby-lang.com/en/)完成的。这些插件实际上就是基于Ruby Gems完成的[①]。为了能写出更有效的Gems，应该了解有关Ruby的相关知识。

在本节中，将会开发自己的输入Input插件，它会从Web服务器中获取数据。该插件的完整代码可以在如下的Github上获取到：https://github.com/kravigupta/mastering-elastic-stack-code-files/tree/5.1.1/Chapter07/logstash-input-weather。这个练习的目的是：学习怎样生成和个性化定制那些能服务于特定目标的插件；同时，通过这个例子，也便于了解插件的结构。

如果能分析出使用了Elastic Stack组件的工程的具体做法，就会发现每一种类的可用

① 译者注：Ruby Gems是对Ruby组件打包的系统。

插件数量很多,其中 Logstash 起到了巨大的作用。如果看到一种新的输入数据源,而且需要从中搜集数据,之后将数据放到 Elasticsearch 或者其他目的地中,就需要新生成一个 Logstash 的插件,用它来帮助搜集相应的数据并完成索引。类似地,如果有复杂的某种数据输入,就可能需要过滤插件或者编解码插件,以便能分隔这些数据的语义块(chunk)并索引它们。类似地,Elasticsearch 可能也不是唯一的数据输出目的地。例如,要捕获特殊的 tweets,并将它们存入数据库中,以便某个第三方应用程序能从该数据库中直接读取这些 tweets 数据。除此之外,还可以找到很多其他类似应用的场景。你会意识到,某种 Logstash 插件可构建适合某种特定应用场合的数据管道(pipeline)的处理流程。

Logstash 使用如下语法生成一个插件:

```
.bin/logstash-plugin generate --type plugin-type --name plugin-name --path plugin-path
```

在上述命令中,需要提供插件类型、插件名称、想要生成插件的路径等信息。

对于插件类型,可以从如下几种类型中接收输入信息:输入插件 Input、过滤器插件 Filter、编解码插件 Codec、输出插件 Output。在本节中,将生成一个输入类型的插件,它用来从 https://openweathermap.org/中获取数据。简便起见,将使用这个插件来获取某时刻某个城市的天气数据信息。

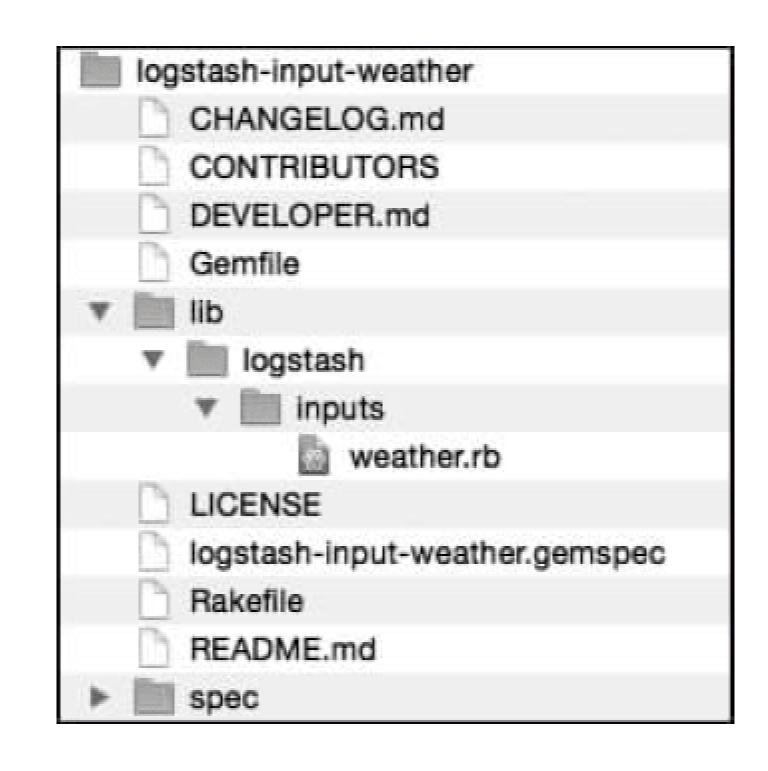
下面的语句可以生成名为 weather 的 Logstash 的输入类型插件。

```
.bin/logstash-plugin generate --type input --name weather --path ../my-plugins/
```

上述语句能生成一个 my-plugins 文件夹,在这里,Logstash 被解压执行。该命令会自动生成 logstash-input 作为插件名字,并在 my-plugins 文件夹中生成一个名为 logstash-input-weather 的文件夹。生成的插件是一个简单的基于 Ruby gem 的项目。

1. 剖析一个插件

从 Git 上检出(check out)插件 plugin,其文件夹结构类似下图所示。



一个 Logstash 插件包括如下组成部分：

- 有一个名为 gemspec 的、用于描述该工程的文件，见上图中的 logstash-input-weather.gemspec 文件。
- 在 lib/logstash/inputs 文件夹下有一个 rb 文件(图中所示为 plugin-name.rb)，这里列出了生成的插件代码，它包含一个有关插件文件的实现方法的示例。
- 文件夹 spec(参见上图)包含所有的针对该插件的测试。
- 其他相关文件用于帮助基于 Ruby Gem 的代码构建，还有一些是相关的说明文档，例如，Rakefile、README.md 等。

2. weather.rb 文件

这是一个主要的文件，含有实现该输入插件的主要处理逻辑。该插件类名为 weather，它扩展自 Logstash::Inputs::Base 类。在这个插件类中的代码示例生成了一个重复事件 event。它有两个关键方法：

- run(queue)：包括设计的实际的处理逻辑。在 run 方法中有一个 while 循环，它只在出现 stop 符号标记时才停止。该循环在设定的时间间隔(默认为 1 秒)会静默。这种方法将队列作为参数。准备好一个事件时，会将该事件推送到这个队列中去等待执行。
- stop()：当 Logstash 关闭时，调用此方法。

3. 插件的逻辑实现

在开始设计实现插件前，需要从 https://openweathermap.org 获得 API key，它将被用来通过 weather API 得到城市的相关数据。在该网站上注册后，打开 https://home.openweathermap.org/api_keys，为你的使用生成一个相关的 API key。

我们将要使用的这个 API 调用，可以用来获取城市的当前数据信息。为了得到更详细的 API 及其相应的返回信息，可在 https://openweathermap.org/current 得到相关内容。给定城市名 cityName 和 API key 值等，相关信息可以在如下地址找到：http://api.openweathermap.org/data/2.5/weather?q=cityName&appid=key。

使用时，需要用感兴趣的城市名替换 cityName，且用得到的 key 值替换 API Key。

针对该 Web 服务的输出是 JSON 对象，含有当前某个城市的天气信息的一些细节内容。一个 JSON 返回的结果类似如下：

```
{
    "coord": { "lon": 72.62, "lat": 23.03 },
    "weather": [
        { "id": 800, "main": "Clear",
        "description": "clear sky", "icon": "01d" }
```

```
    ],
    "base": "stations",
    "main": {
        "temp": 300.15, "pressure": 1014, "humidity": 18,
        "temp_min": 300.15, "temp_max": 300.15
    },
    "visibility": 6000,
    "wind": {
        "speed": 2.6, "deg": 30
    },
    "clouds": { "all": 0 },
    "dt": 1486643400,
    "sys": {
        "type": 1, "id": 7758, "message": 0.0129, "country": "IN",
        "sunrise": 1486604711, "sunset": 1486645354
    },
    "id": 1279233,
    "name": "Ahmedabad",
    "cod": 200
}
```

上述返回信息是针对印度 Ahmedabad 的气象数据，涉及温度、日照时间、下雨、风力和城市元信息(如国家、地理位置等)。

下面完成自己定制插件的开发，它能抽取所有的字段信息并将其存储在索引中。下面是需要理解的编写插件的主要步骤。

1) 从 API 端点读取数据

为了能够从 openweathermap.org 上获取信息，需要调用 Web 服务端点 endpoint。为此，需要引入 net/http 库。在 weather.rb 文件最开头的部分增加下面这行信息：

```
require "net/http" #for calling web service url
```

应该在变量中定义 API URL。在这之前，需要定义 cityName 和 key，并根据 Logstash 配置文件中的需要给定其参数。按照如下方式定义这些变量：

```
config :cityName, :required =>true, :default =>""
config :key, :required =>true, :default =>""
```

对上述两个参数，没有设定默认值，希望用户能够通过使用 Logstash 配置文件为该数据管道 pipeline 提供相应的值。也可以按照需求设定这些配置信息。如果不设定这些值，该插件将不会启动收集数据。现在，可以按照如下方式设定 URL。

```
"http://api.openweathermap.org/data/2.5/weather?q="+cityName+"&appid="+key
```

现在,可以通过下面的方式来完成一次 Web Service 调用。

```
response =Net::HTTP.get_response(URI.parse(url))
```

如果数据调用正确,就可以在 response.body 中得到想要的数据,并可以使用下面的方法完成对 JSON 格式数据的解析。

```
weatherData =JSON.parse(response.body)
```

weatherData 变量将会存储想要索引的 JSON 格式的信息。

2）准备事件

一旦在相应变量中得到了天气数据,就能使用如下的 Logstash::Event.new( )方法来准备事件 event。

```
event =LogStash::Event.new( )
```

Event.new( )能用字段作为参数。如果在事件初始化时向 cityName 中设置数据,可采用如下的调用方式:

```
event =LogStash::Event.new("cityName" =>cityName)
```

现在,能使用 event.set()方法来向事件 event 中增加更多的字段。可考虑如下方法:

```
event.set("cityName", cityName)
```

上述方法的返回值是含有 JSON 对象和数组字段,其值是诸如数值型或字符串类型的简单数据类型。为将 JSON 返回值转换到多个字段中,需定义拥有字段名称的几个变量。

```
config :jsonFields, :default =>"main, clouds, sys, coord, wind"
config :arrayFields, :default =>"weather"
```

这些字段也能增添到配置文件中,其默认值的设置和字段名一样。有时,想利用数据管理 pipeline 配置来改变字段名称,这样做是可以的,只是这些字段未来可能要被拆分。

请看如下循环的例子:

```
jsonFieldsKeys =@jsonFields.split(",")
arrayFieldsKeys =@arrayFields.split(",")
weatherData.each do |k,v|
    if(jsonFieldsKeys.include ?k)
      v.each do | key, val |
        event.set(k.to_s +"_" +key.to_s, val)
      end
```

```
    else
      if(arrayFieldsKeys.include ?k)
        v.each do | obj |
        obj.each do | key, val |
          event.set(k.to_s +"_" +key.to_s, val)
        end
      end
    end
    event.set(k.to_s, v)
    end
end
```

上述命令将使得所有期望的字段值被拆分并存储。

3）发布事件 event

通过下面列出的 decorate 方法处理事件 event。如果有任何定义标记 tag，则它们是可能添加到相应事件中的。

```
decorate (event)
```

完成 decorating 方法后，把处理好的事件 event 放入队列中，以便将其发布到 Elasticsearch 中。

```
queue <<event
```

到此，结束个性化的逻辑处理，它们驻留在 while 循环中并进入队列排队等待运行。这个循环以某个内部设定的时间变量（默认 1 秒）静默。在代码实现方面，已经完成了具体逻辑任务的处理。现在，可以开始构建插件了，该插件将会在 Logstash 中安装，并在 Kibana 的数据流中测试。

> Openweathermap.org 网站允许每小时最大 60 个访问需求，因此，在插件中可以设定 60 秒的访问时间间隔。

4. 构建与安装插件

一旦完成了插件开发，就能以 Ruby Gem 构建插件，它以后就能像其他插件一样安装到 Logstash 中以供使用。为了构建一个插件，可以使用 Gem 命令：

```
gem build logstash-input-weather.gemspec
```

上述命令运行后，如果得到类似这样的错误信息：

```
ERROR: While executing gem ... (Gem::InvalidSpecificationException)
```

```
"FIXME" or "TODO" is not a description
```

就应该用正确的内容来更新 gemspec 文件了。一般情况下,需要修改位于 logstash-input-weather. gemspec 文件中的摘要、描述、主页的值。如果没有任何错误输出,就能够得到类似下面的输出信息:

```
Successfully built RubyGem
Name: logstash-input-weather
Version: 0.1.0
File: logstash-input-weather-0.1.0.gem
```

这将会生成一个以.gem 为扩展名的、位于插件 plugin 文件夹中的插件文件。使用下面的 **bin/plugin** 命令,可以将其安装到 Logstash 中。

```
.bin/logstash-plugin install /path/to/logstash-input-weather-0.1.0.gem
```

现在,就可以利用这个插件从 openweathermap. org 中获取数据了。为验证该插件是否已经安装好,可以通过运行下面的命令来列出所有插件,看看新生成的插件名是否能够显示出来:

```
./bin/logstash-plugin list
```

5. 测试插件

插件安装好后,可以通过 Logstash、Elasticsearch、Kibana 的组合来测试它。首先,为 Logstash 产生一个名为 weather. conf 的配置文件,它用于从 opeaweathermap. org 中搜集数据并将数据发送到 Elasticsearch 中。简便起见,现在不用任何过滤插件 filter。这个简便的配置文件内容类似如下形式:

```
input {
    weather {
      key => "<your-api-key-here>"
      cityName => "Ahmedabad"
    }
}
output {
      elasticsearch {
          index => "weather"
          document_type => "citiesData"
      }
}
```

实际中,常常需要调试这个插件。可以设置 log. level(在 Logstash-5. 1. 1/config/

logstash.yml 中)，以 debug 形式启动 Logstash。

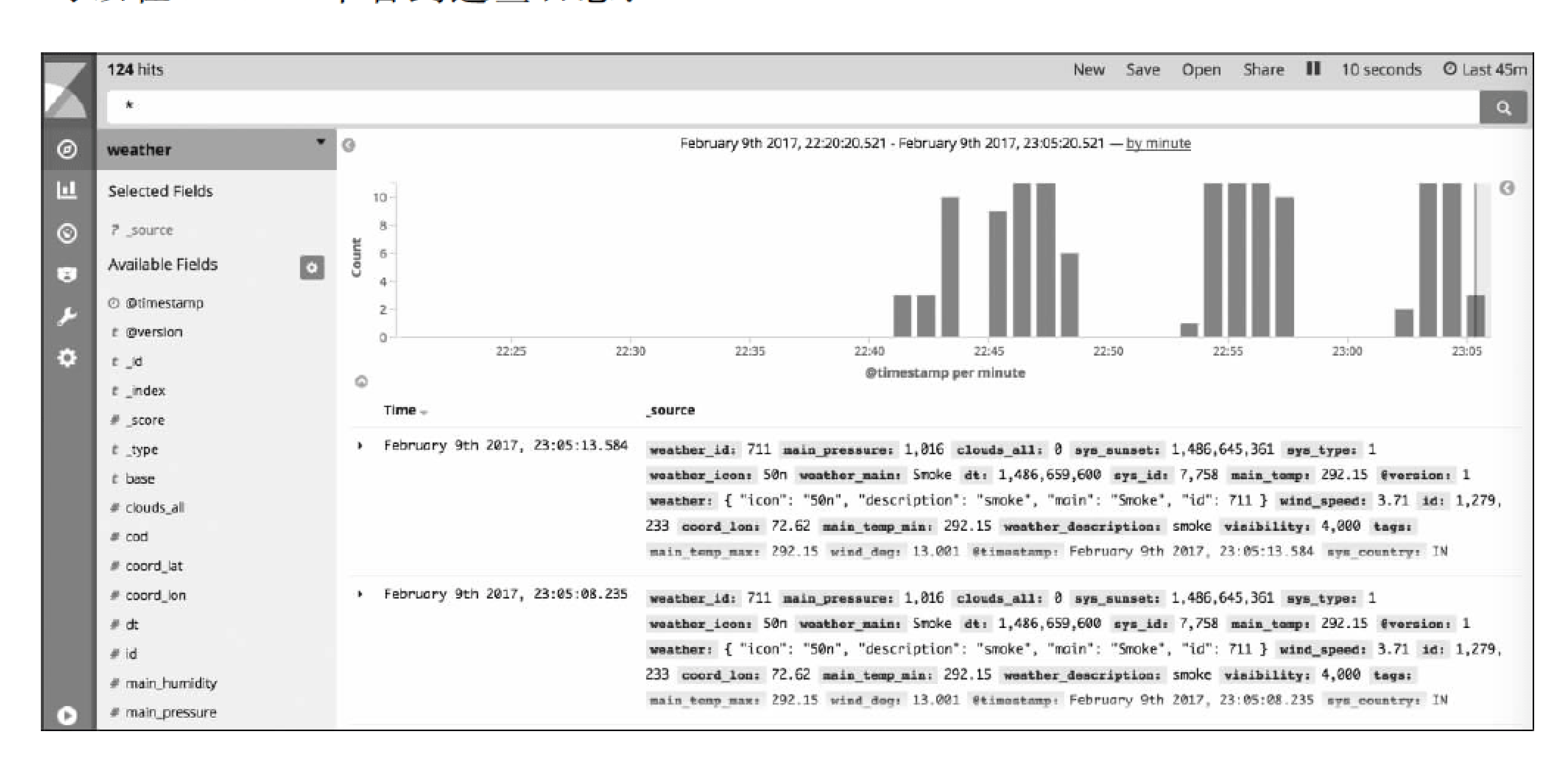
```
log.level:debug
```

另一件需要完成的工作是在数据处理管道 pipeline 的输出部分增加以 stdout 方式输出。它将会在控制台显示生成了什么样的事件 event，以便于快速分析、处理。

用自己的 API key 替换<your-api-key-here>占位符，可以使用如下的方式运行 logstash。

```
./bin/logstash -f conf/weather.conf
```

可以在 Kibana 中看到这些日志：



如果以 debug 调试模式运行 Logstash，就会在控制台上看到类似如下的信息：


```
[2017-02-09T23:07:13,394][DEBUG][org.apache.http.wire] http-outgoing-0 >>
"{"weather_id":711,"main_pressure":1016,"clouds_all":0,"sys_sunset":1486645
361,"sys_type":1,"weather_icon":"50n","weather_main":"Smoke","dt":148665960 0,"
sys_id":7758,"main_temp":292.15,"@version":"1","weather":[{"icon":"50n",
"description":"smoke","main":"Smoke","id":711}],"wind_speed":3.71,"id":1279
233,"coord_lon":72.62,"main_temp_min":292.15,"weather_description":"smoke",
"visibility":4000,"tags":[],"main_temp_max":292.15,"wind_deg":13.0008,"@tim
estamp":"2017-02-09T17:37:13.373Z","sys_country":"IN","sys_sunrise":1486604
705,"coord_lat":23.03,"name":"Ahmadabad","cod":200,"main_humidity":37,"sys_
message":0.1309,"base":"stations"}[\n]"
```


能看到事件 event 提交后的日志，能用文档类型 type 验证日志(见上图 Kibana 中的_type)。这表明我们的插件是正常工作的，能搜集数据，也能发送数据到 Elasticsearch 完成索引。

类似地，也能编写其他类型的插件。前面已经提到，拥有 Ruby 编程知识将有助于开发

Logstash 插件。

> 在 12 章案例分析——Meetup 中，也为 meetup.com 生成了一个输入插件。这个插件比上述例子要复杂一些。

7.3　扩展 Beats

从前面章节已经看到，针对 Elastic 团队，有四种核心 Beat，即：Filebeat、Packetbeat、Metricbeat 和 Winlogbeat。单击下面的链接，可以显示庞大的 Beats 家族成员：

```
https://www.elastic.co/guide/en/beats/libbeat/5.1/community-beats.html
```

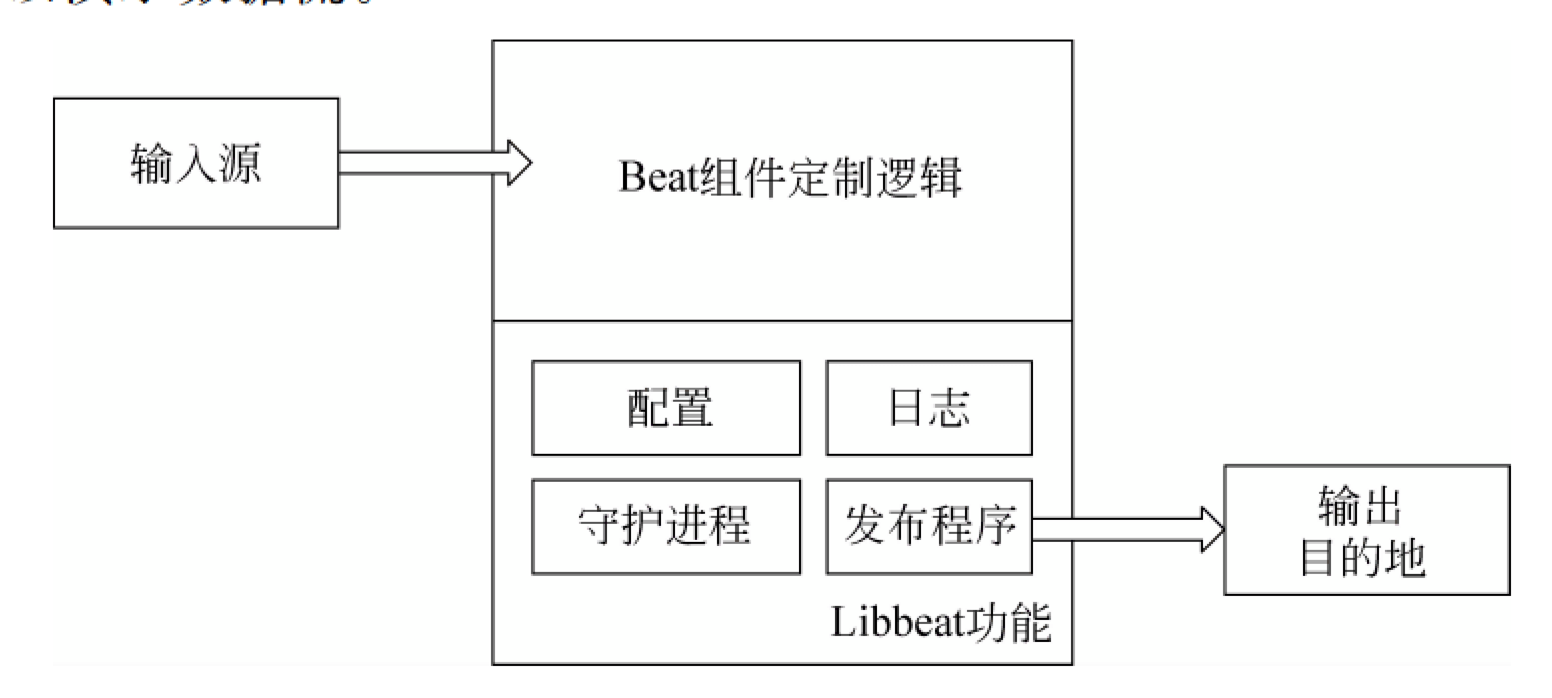
所有这些 Beats 都是构建在 Libbeat 架构上的。有些情况下，可能找不到一个能满足个性化需要的 Beat，需要生成一个新的 Beat，这个新生成的 Beat 也遵循同样的处理架构，只需关注和定制自己的处理逻辑（即：如何从数据源取出数据，如何准备一个事件来把数据发送到 Elasticsearch 或 Logstash 中）。

在本节中，我们将会看到如何通过运用 Libbeat 架构生成一个 Beat，用于最小化对 Logstash 的使用。对于 Logstash 插件，也用同样的例子。我们将会从 Web 服务器上读取数据，并在 Elasticsearch 中完成索引。对于所有生成的 Beat 代码，作为参考，可以单击如下的 Github 链接：

https://github.com/kravigupta/mastering-elastic-stack-code-files/tree/5.1.1/Cha pter07/weatherbeat

7.3.1　Libbeat 框架

为了理解 Libbeat 架构是如何生成 Beat 的，现在看一个 Beat 例子。它从一个应用中搜集基础数据（多数情况下是静态的数据），之后将数据发送到 Elasticsearch 或 Logstash 中。下面的图示可以演示数据流。



如果分析这个图表，就会发现其中一些个性化定制的从应用中读取数据的处理逻辑，用来完成解析数据、准备事件等工作。对不同的 Beat，处理是不用的，因为从应用中获取数据有赖于程序提供了什么样的渠道。对每一个 Beat，各自解析获得的数据的方式可能是不同的。最后，部分过滤数据将会被用于准备事件。这是个性化定制逻辑中的一部分。作为 Beat 开发者，应该注意这些特殊的部分。

除此之外还有一些通用性的任务。其中之一是发布者，用于把准备好的事件发送到配置好的输出管道中。这里还有其他对每个 Beat 很有必要的一些任务要做，它们都在 Libbeat 架构中提供(可将它们认为是一个帮助库，用来提供最通用的所有 Beat 都需要的包文件 package)。下面简单列出一些组件。

- **配置 Configuration**：记住，有一个针对所有 Beat 文件的配置信息文件(filebeat. yml、packetbeat. yml 等)。类似地，也有一个针对新生成的 Beat 文件的类似 my-custom-beat. yml 的配置文件。它有助于 Beat 用户完成设置，例如从哪里获取信息，主机 hostname，端口号和授权相关的细节信息以及数据将要发送到的目的地等。
- **发布者 Publisher**：个性化定制的逻辑部分在准备一个事件时，需要向发布者 Publisher 发送事件日志 log。这个发布者是由 Libbeat 架构提供的，用于读取配置信息并将数据发送到各自的目的地。
- **日志 Logging**：每一个好的应用程序都要有一个日志管理机制。开发者需要属性 log4J 库等日志管理工具，它们能在 Java 应用程序中为我们提供日志处理方面的帮助。Libbeat 架构为 Beats 提供日志，不需要为此写特别的代码。
- **守护进程 Daemon**：多数情况下，可能想以后台服务或者守护进程 Daemon 的方式来运行 Beat，以便利用操作系统为此提供的一些便利(如自举系统、提供事件日志等)。对于自定义的 Beat，Libbeat 有内置的对 Deamons 的支持。对于以服务方式运行 Beat，几乎无须做什么事情。

下面将要学习如何按照我们的选择生成一个 Beat。除了那些(少量的)个性化处理逻辑需要自己编写之外，这种生成的 Beat 具有几乎所有其他相关的部件。

7.3.2 创建一个 Beat

Beat 可以从任何地方(可以是一个进程、一个应用程序、一个程序中的组件、一个网页、一个信息聚合 RSS feed，等等)来输入数据[①]。只要是数据是有用的，就值得分析。

在前面的章节中，为 Logstash 生成输入插件时，是使用 openweathermap. org API 来获

---

① 译者注：RSS feed 是一种基于 XML 技术的聚合标准，是要获取消息的网站内容的种子。获得了 RSS feed 并添加到你的 RSS 聚合器中，RSS 聚合器就会定时按照 RSS feed 解析相应网站上的内容。

取一个城市的数据并作为例子的。为了利用同样的 API 服务参数 endpoint[①]，我们来生成一个 Beat。

假设已经从 openweathermap.org 获得了用于读取数据的 API key。利用它，就可以生成一个 Beat。按如下几个简单步骤来生成一个 Beat。但在这之前，需要安装一些相关的依赖 dependencies（多数情况下，它们是生成 Beat 需要使用的几个相关工具）。

- **Go 语言**：大多数 Beat 使用 Go 编程语言，这是一个开源的语言，可在 https://golang.org/获取。下载后，按照步骤，可以在计算机上安装它。Windows 和 Mac OS X 系统提供安装程序，不必做任何额外的设置操作（包括如何将 Go 纳入 PATH 变量中等）。对于 Linux 环境，需要在安装 Go 的过程中设置 PATH 变量，方法如下：

```
export PATH=$PATH:/usr/local/go/bin
```

  如果个性化定制了安装 Go 的位置，就需要按照如下方法进行设置：

```
export GOROOT=$HOME/installations/go
export PATH=$PATH:$GOROOT/bin
```

个性化定制了安装路径后，GOROOT 一定要设置。也应该设置 GOPATH 变量，它用来指向存放工程的工作空间 workspace。

```
export GOPATH=$HOME/workspace
```

- **Python**：可能大家已经对这个强有力的编程语言很熟悉了，它可以在 https://www.python.org 获取。可以从这里找到适合自己应用环境的安装程序包，下载并安装它。
- **Virtualenv**[②]：这个工具可以隔离 Beat 工程，以便在运行 Beat 时不影响其他的工程项目。它可以在这里获取：https://virtualenv.pypa.io/en/stable/。推荐使用 pip，完成安装。

```
$ sudo pip install virtualenv
```

- **获取并安装 Cookiecutter**[③]：这是一个基于命令行的应用程序，可以使用工程模板来生成工程。可以在 Github 上获取它：https://github.com/audreyr/cookiecutter。可以使用 pip、easy_install 或者 brew 完成安装（在 Mac OS X 环境中）。

---

① 译者注：API endpoint 又称“端点”，表示 API 的具体网址。
② 译者注：通过创建独立 Python 开发环境的工具，Virtualenv 可解决依赖、版本以及间接权限问题。
③ 译者注：Cookiecutter 是一款快速建立工程模板的 Python 命令行工具。

```
$ sudo pip install cookiecutter
$ sudo easy_install cookiecutter
$ brew install cookiecutter
```

pip 命令是一个包管理器，用于安装 Python 程序包。如果没有如上这些可用的工具，建议安装其中的某一项。

一旦已经安装了如上这些工具，在生成 Beat、利用生成的 Beat 工作时，就应该不会有什么问题了。请按照如下步骤进行：

(1) 获取 beat generator 和 Libbeat，方法如下：

```
go get github.com/elastic/beats
```

运行上述命令后，可能会得到错误信息："package github. com/elastic/beats: no buildable"。可以不用管它，仍旧可以继续构建 Beat。Go 源文件在这里：/Users/ravi. gupta/workspace/src/github. com/elastic/beats。

该命令用于下载 beat generator 和 Libbeat，并放在如下位置：

```
workspace
    -src
        -github.com
            -elastic
```

(2) Go 编程语言有诸多很方便的特性，其中之一就是使用 Github。如果有 Github 账号，可以使用它登录。在自己的工作空间中，在 github. com 文件夹下，为你的用户名生成一个文件夹，在这个数据仓库下，将来可以存放生成的内容。最终的文件夹结构类似如下形式：

```
workspace
    -src
        -github.com
            -your-user-name
```

在命令行方式下，进入如下文件夹：

```
workspace/src/github.com/your-user-name
```

使用 packt 作为我们的用户名。

(3) 在 workspace/src/github. com/packt 文件夹下，使用 Cookiecutter 生成一个 Beat。

```
cookiecutter $GOPATH/src/github.com/elastic/beats/generate/beat
```

上述命令将会按照我们的选择（Beat 名、包 packet 名等）生成 Beat。下面是以 Weatherbeat 为 Beat 名的一个例子：

```
project_name [Examplebeat]: Weatherbeat
github_name [your-github-name]:packt
beat [weather]:
beat_path [github.com/packt]:
full_name [Firstname Lastname]:
```

(4) 一旦 cookiecutter 已经通过命令的方式运行，就会在 src/github.com/packt/weatherbeat 文件夹下生成 Weatherbeat。这个 Beat 是一个没被加工的模板，需要继续在上面完成一些工作。首先，需要获取必要的依赖 dependency。进入到 beat 文件夹 weatherbeat，运行如下 make 命令：

```
cd weatherbeat
make setup
```

运行 make 后，将会为我们的工程增加依赖 dependency。现在，这个 Beat 已经准备好了，但尚未有个性化定制逻辑部分（即从 openweathermap.org 中获取一个城市的天气信息）的代码。

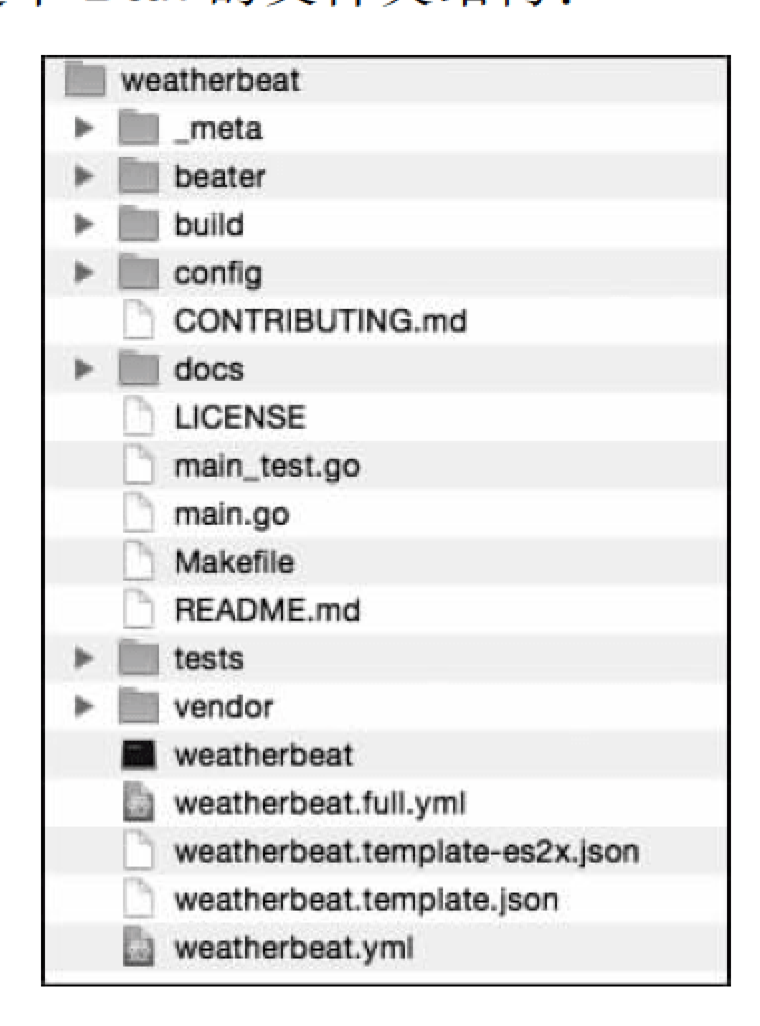
(5) 最后一条命令用于构建 Beat 源码，输出一个可以运行的文件。在相同的文件夹下，执行如下命令：

```
make
```

这将会为我们的 Beat 生成一个可执行文件。

7.3.2.1　剖析 Beat

到此为止，我们生成的 Beat 已经具有了除获取一个城市气象信息这个处理逻辑以外的所有内容了。我们来看一看这个 Beat 的文件夹结构：



一个 Beat 应用由如下部分组成：

- 执行部分 executable：
  - ◆ 在这里例子中，weatherbeat 是用于运行 Beat 并从 openweathermap.org 中获取数据的执行文件。它将数据发送到 Elasticsearch 或 Logstash 中。
- 模板 template 和配置文件 configuration file：
  - ◆ 每个 Beat 都有一个＜beat＞.yml 文件和一个＜beat＞.full.yml 文件，含有针对该 Beat 的模板配置文件。
  - ◆ 在＜beat＞.template.json 文件中有 mapping 模板。
- 实体点 entry point 和个性化的处理逻辑：
  - ◆ main.go 文件是源码中的入口点 entry point，它内部调用 beater/＜beat＞.go 文件，后者用于实现 Beater 界面。

> ℹ每一个 Beat 都需要实现 Beater 的界面 interface，而 interface 是由 Libbeat 架构所提供的。

- 位于 docs 文件夹下的说明文档(如 README 等)。
- 位于 test 文件夹下的测试文件。
- 必要的项目依赖文件和构建助手。
  - ◆ 在 vendor 文件夹下，有一个完整的 Beat 包，它用于为生成的 Beat 提供项目依赖支持。
  - ◆ 在 build 文件夹下有必要的用于构建 Beat 的工具。

7.3.2.2 Beat 配置

对于每个生成的 Beat，在＜beat＞.yml 文件中均有一些可用的配置。在这里，weatherbeat.yml 文件含有这些配置。为操作简便，默认输出到 Elasticsearch 及相应周期(时间间隔)的属性均是打开的(enable)。

```
weatherbeat:
  #Defines how often an event is sent to the output
  period: 5s
  output.elasticsearch:
   #Array of hosts to connect to.
   hosts: ["localhost:9200"]
```

我们的 Beat 使用周期 period 作为时间间隔，周期性地把事件发送到 Elasticsearch 中。这里还有其他一些默认运行的配置，用于记录日志，控制 Logstash 输出，设置 tags、shipper

名字等。

> 默认情况下,向文件输出日志的属性是打开的(enable),因此日志不会在控制台上输出。为了使增加的日志为有效状态,可以使用 logging.to_syslog:true。

所有的完整配置和相关文档均可在 weatherbeat.full.yml 中找到。

7.3.2.3 weatherbeat.go 文件

每一个 Beat 都被使用如下的方法设置为类型 type:

```
type weatherbeat struct {
    done chan struct{}
    config config.Config
    client publisher.Client
}
```

这个 Beat 使用 weatherbeat.yml 中的 New()方法来读取相应的配置信息。

```
func New(b *beat.Beat, cfg *common.Config) (beat.Beater, error) { ... }
```

由于该文件是 Beat 功能驻留和实际执行部分,所以它是全部 Beat 文件中最重要的一个文件。该文件应基于 Beat 命名。本例中,该文件被命名为 weatherbeat.go(位于 beater 文件夹下)。

它实现了 Beater 界面功能。Beater 界面有两个必须要实现的方法:

- Run (b *Beat) error:该方法含有实际的处理逻辑,当 Beat 启动时自动运行。
- Stop ( ):当相应的 Beat 停止时(使用 Ctrl+C 命令),调用此方法。

如果打开 weatherbeat.go 文件,会发现上述两个方法已经实现了。Run( )方法在每个 n 秒的时间间隔(该阈值在 weatherbeat.yml 中定义),生成一个事件,并将该事件提交给发布者(它用于发布该事件到预先定义好的目的地,而该目的地是在配置文件中已经定义好的)。

7.3.2.4 实现 Beat 的逻辑

现在该轮到处理我们定制的逻辑部分了。它是 weatherbeat 的 Run()方法中的一部分。在内核中,需要从 API 服务参数 endpoint 读取数据,按照某种适当格式解析它,准备事件,并完成发布。

API 服务参数 endpoint 可以从如下方式得到:

```
http://api.openweathermap.org/data/2.5/weather?q=cityName&appid=key
```

下面看看实现插件的主要步骤。

1. 增加配置

我们需要从用户那里得到cityName和API key。在weatherbeat.yml配置文件中，增加如下两个配置信息：

```
weatherbeat:
    cityName: ""
    key: ""
```

之后，运行如下命令：

```
make update
```

现在，完成了对位于配置文件夹下的配置文件config.go的更新。配置文件的架构类型看起来是下面这样的：

```
type Config struct {
    Period time.Duration config:"period"
    CityName string config:"cityName"
    Key string config:"key"
}
```

已经增加了两项配置：CityName和Key，它们都能在beat中应用。

2. 从API读取数据

需要导入(import)net/http包，用于读取Web服务器中的数据。需要导入io/ioutil包来完成读取输入数据的工作。

首先，在Run函数中，为cityName[①]和key生成两个变量：

```
cityName:=bt.config.CityName
key :=bt.config.Key
```

由于已经在Config架构中增加了它们，因此能通过使用bt.config得到数据。之后，使用http.Get()方法完成调用：

```
url :=
"http://api.openweathermap.org/data/2.5/weather?q="+cityName+"&appid="+key
  response, error :=http.Get(url)
  defer response.Body.Close()
  body, error :=ioutil.ReadAll(response.Body)
```

① 译者注：原文为city。

现在,数据体将会包含输入数据。如果出现问题,会在 error 变量中得到相关的错误提示信息。

3. 解析数据

数据体变量(可以是数组或字节的形式,它实际上应该是一个转换成字符串的 JSON 数据块)包含的是 Web 服务调用的输出数据。在 Go 语言中,使用 Unmarshaling 方法将一个字符串或字节转换为 JSON 格式。为完成这个任务,需要设置一个未来将要被执行映像 mapping 的类型 type,它使用关键词架构定义。对于每个 JSON 响应,类型 type 的定义如下:

```
type WeatherDataTypes struct {
    Coord CoordTypes
    Weather [ ]WeatherTypes
    Base string
    Main MainTypes
    Visibility int
    Wind WindTypes
    Clouds CloudTypes
    Dt int
    Sys SysTypes
    Id int
    Name string
    Cod int
}
type CoordTypes struct {
    Lat float64
    Lon float64
}
type MainTypes struct {
    Temp float64
    Pressure int
    Humidity int
    Temp_min float64
    Temp_max float64
}
type WindTypes struct {
    Speed float64
    Degree int
}
type CloudTypes struct {
```

```
    All int
}
type SysTypes struct {
    Id int
    Message float64
    Country string
    Sunrise int
    Sunset int
}
type WeatherTypes struct {
     Id int
     Main string
     Description string
     Icon string
}
```

这些结构类型是在 Run()功能外定义的。使用主类型 WeatherDataType，能将 JSON 返回值转换为一个适当的结构：

```
Message :=(*json.RawMessage)(&body)
var weatherData WeatherDataTypes
json.Unmarshal(*Message, &weatherData)
```

现在，weatherData 变量将拥有从 JSON 来的全部字段，它们可以被用来做进一步处理，以便生成事件 event。上述工作需要导入(import)encoding/json 数据包。

4. 准备事件 event

使用 common. MapStr，准备一个能发送到发布者 Publisher 的 JSON 对象类型 type。对于事件 event，只是把 sunrise、sunset、temperatures 增加到这里(参见下面的代码)。添加了所有字段的 Beat 可在这里获取：https://github. com/kravigupta/mastering-elastic-stack-code-files/tree/5. 1. 1/Cha pter07/weatherbeat。

```
event :=common.MapStr{
    "@timestamp":    common.Time(time.Now()),
    "type":          b.Name,
    "sunrise":       weatherData.Sys.Sunrise,
    "sunset":        weatherData.Sys.Sunset,
    "temp":          weatherData.Main.Temp,
}
```

> ℹ每一个事件对象 event object 应有一个时间戳 timestamp(一般是当前时间)、一个类型 type(设置为 Beat 名称)。

5. 发布事件 event

最后，使用下面的方法发布事件 event：

```
bt.client.PublishEvent(event)
```

这将会向发布者 Publisher 发送事件对象 event object。Publisher 用于向已经定义好的目的地发送相关内容。

生成的 Beat 将以类型下面的方式被执行(只能调用 Run()方法)。

```
func (bt * Weatherbeat) Run(b * beat.Beat) error {
     logp.Info("weatherbeat is running! Hit CTRL-C to stop it.")
     bt.client =b.Publisher.Connect()
     ticker :=time.NewTicker(bt.config.Period)
     for {
       select {
       case <-bt.done:
         return nil
       case <-ticker.C:
       }
       cityName:=bt.config.CityName
       key :=bt.config.Key
       url :=
   "http://api.openweathermap.org/data/2.5/weather q="+cityName+"&appid="+key
       fmt.Println("URL is " +url)
       resp, err :=http.Get(url)
       if err !=nil {
           fmt.Println("Something went wrong")
       }
       defer resp.Body.Close()
       body, err :=ioutil.ReadAll(resp.Body)
       Message := (* json.RawMessage)(&body)
       var weatherData WeatherDataTypes
       json.Unmarshal(* Message, &weatherData)
       event :=common.MapStr{
         "@timestamp": common.Time(time.Now()),
         "type": b.Name,
         "sunrise": weatherData.Sys.Sunrise,
         "sunset": weatherData.Sys.Sunset,
         "temp": weatherData.Main.Temp,
       }
       bt.client.PublishEvent(event)
```

```
        logp.Info("Event sent")
    }
}
```

需要导入的 import 部分如下：

```
import (
    "fmt"
    "time"
    "io/ioutil"
    "net/http"
    "strconv"
    "github.com/elastic/beats/libbeat/beat"
    "github.com/elastic/beats/libbeat/common"
    "github.com/elastic/beats/libbeat/logp"
    "github.com/elastic/beats/libbeat/publisher"
    "github.com/packt/weatherbeat/config"
)
```

7.3.2.5 运行 Beat

现在，前期准备工作已基本完成，还需要使用 make 方法来构建 Beat。如果 Beat 源发生了变化，则需要使用 make 来再次构建 Beat。到 Beat 文件夹下，用如下命令来运行 beat。

```
./weatherbeat
```

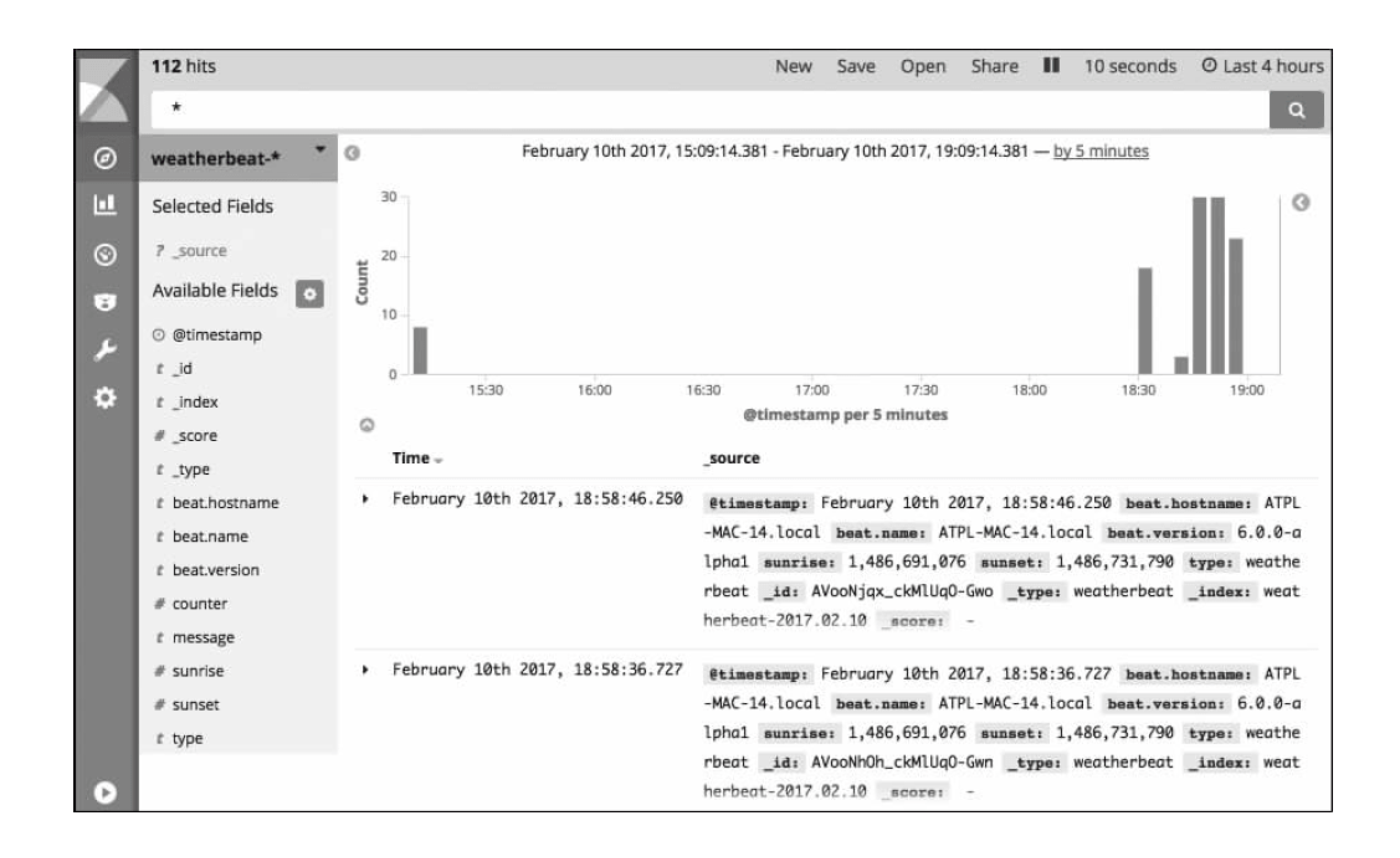
现在，Beat 将会在设定的时间开启搜集数据的工作，并将数据发送到 Elasticsearch。在 Elasticsearch 的日志中，会找到类似这样的信息：

```
[2017-02-10T15:11:54,839][INFO ][o.e.c.m.MetaDataMappingService] [A5S5Dxl]
[weatherbeat-2017.02.10/9tO1oFBwTzePDBEC7YXctA] create_mapping
[weatherbeat]
```

它将会尝试为索引 weatherbeat 生成映像 mapping(如果不存在)。也能在 Kibana 检查发送到 Elasticsearch 的数据。在 Kibana 的 discover 标签页中，选择 weatherbeat-* 索引模式，会看到由我们生成的 Beat 所发送的日志。

从下图能发现，这里有一些由 Beat 生成的日志事件(log events)，表明构建的 Beat 是可以正常工作的。通过在 Kibana 中使用不同的方法，能够进一步使用数据来完成可视化。

> ℹ 默认情况下，weatherbeat 的索引名可能是：-weatherbeat-%{+yyyy.MM.dd}。如果想改变它，可以在 weatherbeat.yml 配置文件中修改索引设置。



7.4　扩展 Kibana

搭建 Elastic Stack 环境时，做一些必要的修改，可使它能满足个性化的要求。在大多数情况下，相比于 Stack 中的其他组件，对 Elasticsearch 所做的更改是比较小的。作为 Stack 组件中面向最终用户的 Kibana 部分，其变化往往是最多的。如果没有什么复杂需求，客户端 Client(作为一个服务提供者时)希望能自定义用户的接口 UI，以满足公司对前端设计的个性化需求。

Elastic Stack 提供各种各样的方式，满足定制和扩展 Kibana 的需求(可以写 Kibana 插件，创建新的可视化组件，修改源代码等)。要做到这些，必须要熟悉 HTML、CSS、Javascript、Node.js、AngularJS 以及相关的用于图表和可视化的库。Kibana 基本上是基于 Node.js 的应用程序。了解 Node.js，将有助于了解和定制 Kibana。

本节中将开发一个改变 Kibana UI 外观的插件。为简单起见，只在运行时修改 Kibana 每个页面的标题。通过这个例子可看到开发这样的插件是多么容易的一件事。如果你想检出(check out)相关的源代码，可以在如下的 Github 上得到。https://github.com/kravigupta/mastering-elastic-stack-code-files/tree/5.1.1/Chapter07/branding-plugin。插件的名称可定制，它可以作为样例来定制 Kibana。所有已知的 Kibana 插件都在这里列出：https://www.elastic.co/guide/en/kibana/5.1/known-plugins.html。

在开始后续工作前，建议设定好 Kibana 开发环境。可从 Github 上克隆 Kibana 的源代

码，CONTRIBUTING.md 文件包含设置开发环境的所有步骤。

7.4.1 设置 Kibana 开发环境

为了设置开发环境，应该从 Github 上析出 Kibana 工程，并将其克隆到你的机器上。

```
git clone https://github.com/{username}/kibana.git
```

定位到最新克隆的 Kibana 源代码，切换到将要使用的分支上。这里，使用的是 5.1 版本（生成了一个本地分支），例如 kibana-local，指向各自的分支，如下所示：

```
$ git checkout -b kibana-local origin/5.1
```

安装 Kibana 需要的节点版本。

```
nvm install "$ (cat .node-version)"
```

> 如果没有安装 nvm，可以使用下面的命令来完成：curl-o-https://raw.githubusercontent.com/creationix/nvm/v0.32.1/ install.sh | bash。

Kibana 5.1.X 版本使用 Node.js v6.9.0，可根据操作系统环境，从 https://nodejs.org/en/download/releases/ 直接下载相应版本的 node.js。为了能直接生成插件，应该确保 Kibana 使用的 node.js 版本与此一致。

之后，需要安装节点 node 的依赖 dependency：

```
npm install
```

Kibana 将会从 Elasticsearch 中读取数据。应确保运行的 Kibana 和 Elasticsearch 为同样的版本。在开发模式下，最好只从开发环境中安装和运行 Elasticsearch。如下的命令可以运行 Elasticsearch。

```
npm run elasticsearch
```

如果不想使用这个 Elasticsearch 的实例 instance，则可以像 Kibana 一样，运行自己的 setup。

一旦 Elasticsearch 可用，就完成了 setup。可以运行 Kibana 开发服务。

```
npm run
```

在默认状态下，这将会在 https://localhost:5601 运行 Kibana（基于 HTTPS 协议）。如果收到错误信息（SSL in logs），则可在配置中提供 Kibana 的证书，或者以非 SSL 方式运行 Kibana。

```
npm run -- --no-ssl
```

上述这些操作，在任何情况下（或者想写一个插件，或者想修改 Kibana 的核心代码）都是必要的。为了加速插件开发的进程，可以使用 generator-kibana-plugin。这是一个基于 Yeoman① 的 Kibana 插件生成器。

> Yeoman(http://yeoman.io/)是帮助工具，用于加速 Web 应用的开发进程。它能为插件或应用程序生成一个框架。

使用如下命令安装 Yeoman 生成器 generator：

```
npm install -g yo
```

使用如下命令，安装 Kibana 生成插件：

```
npm install -g generator-kibana-plugin
```

现在，已经做好所有准备工作了，可以开始生成插件了。

7.4.2　生成一个插件

一旦搭建起 Kibana 开发环境并安装好插件生成器 generator，就能够生成插件了。现在，在我们克隆 Kibana 源代码的文件夹中，为插件生成一个文件夹。假设插件名字是 branding-plugin：

```
mkdir branding-plugin
```

移动到上述文件夹中，执行以下命令：

```
cd branding-plugin
```

现在，使用 Yeoman 生成插件：

```
yo kibana-pluging
```

执行时，Yeoman 将会询问一些问题（如插件名称、插件描述、插件版本等）。这里需要指定 Kibana 版本：

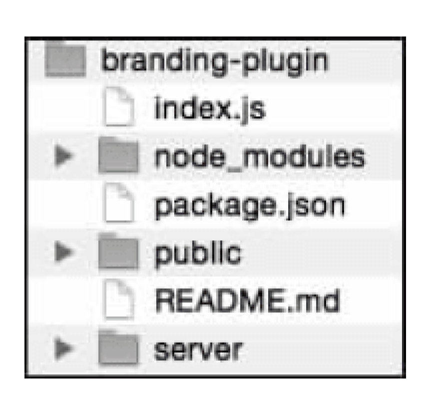
```
? Your Plugin Name branding plugin
? Short Description A plugin to help modify Kibana for branding.
? Target Kibana Version 5.0.0
```

① 译者注：Yeoman 定义了一套用于提高前端工程师效率的规范工作流工具。

该命令执行将会用一些时间。执行后，将会得到插件的架构 skeleton。

7.4.3 剖析一个插件

生成的 Kibana 插件将会有类似下图这样的结构。



该插件由位于 server 文件夹的服务端代码和位于 public 文件夹的 Web APP 相关的代码共同组成。还有一个 node_modules 文件夹，在这里存放着该插件所有需要的依赖 dependency。

每个插件必须有一个 package.json 文件，它包含该插件的细节信息（如示例名字、版本、依赖等）。如果没有这个插件，Kibana 就不会以开发模式运行，也将会由于没有 package.json 文件而得到错误的信息。

对于插件而言关键文件是 index.js，这是一个综合了所有内容的主入口点，类型如下：

```
import exampleRoute from './server/routes/example';
  export default function (kibana) {
    return new kibana.Plugin({
      require: ['elasticsearch'],
      uiExports: {
        app: {
          title: 'Branding Plugin',
          description: 'A plugin to help modify Kibana for branding.',
          main: 'plugins/branding_plugin/app'
        },
        hacks: [
          'plugins/branding_plugin/hack'
        ]
      },
      config(Joi) {
        return Joi.object({
          enabled: Joi.boolean().default(true),
        }).default();
      },
      init(server, options) {
        // Add server routes and initalize the plugin here
```

```
            exampleRoute(server);
          }
        });
      };
```

文件中的这个重要部分定义为 uiExports，它定义了想要用插件完成的事情，以及插件实际包括的内容。

Kibana 提供了各种各样的 uiExports，包括增加一个新的可视化 visualization、设置、导航条、新 APP 等。所有可用的 uiExports 都可以在这里找到：https://www.elastic.co/guide/en/kibana/5.1/development-uiexports.html。这些 uiExports 分为如下三组：

- **Hacks**：想要修改 kibana UI 或者拦截 UI 事件时，可能需要 Hacks。这些类型的插件适用于所有已经安装在 Kibana 中的应用程序。
- **UI Registry provider**：有大量的针对各种特征的 ui/registry，如下表所示：

uiExport 类型	Registry 类型
visTypes	ui/registry/vis_types
fieldFormats	ui/registry/field_formats
spyModes	ui/registry/spy_modes
chromeNavControls	ui/registry/chrome_nav_controls
navbarExtensions	ui/registry/navbar_extensions
settingsSections	ui/registry/settings_sections
docViews	ui/registry/doc_views

在上面的表中，第一列是 uiExport 类型，第二列是相应的每个 uiExport 注册时提供的 registry。

- **应用**：Kibana 支持系统中附加的完整应用。为了增加一个应用，需要 app 或者 appsuiExport 类型。

最常用的 uiExport 类型是 visTypes，它可为 Kibana hack（它能按需修改系统、绑定事件 event 等）生成新的可视化应用。

如果要生成使用简单的可视化部件（如一个水平条或其他类型可视化部件）的 Kibana 插件，则可以使用 visTypesuiExport。

现在再回到 index.js 文件，它是针对插件的入口点。默认情况下，该文件含有载入两个 uiExports-hack 和 app 的代码，如果不需要它们，可以删除 pre-added uiExports。被修改过的 index.js 文件可能像下面这样：

```
export default function (kibana) {
        return new kibana.Plugin({
    uiExports: {
    hacks: [
          'plugins/branding_plugin/hack'
                    ]
        },
    });
};
```

只需要 hacks uiExport，它是一个位于插件 public 文件夹下的 .js 文件中的一部分。为了快速检查这个 hack 是否能正常工作，尝试将一个 log 放到 hack.js 文件中，并检查控制台的 log 输出。如果能显示相应的日志，就能确信该插件能正常工作了。

在进行该项工作前，可删掉已经无用的 server 文件夹。另外，除 hack.js 文件外，在 public 文件夹中已经不再需要什么了。

可以按照需要修改 hack.js，以便使其能以类似下面的方式输出日志信息：

```
console.log('Loading the hack for branding plugin')
```

之后，从插件文件夹下运行服务 server。

```
npm start
```

这将会以开发模式启动服务 server，自动在 Kibana 实例中安装插件。它支持热部署 hot deployment，对插件文件进行修改时，将会显示类似如下的日志信息：

```
[info][optimize] Lazy optimization started
[info][optimize] Lazy optimization success in 14.42 seconds
```

一旦在浏览器中载入 Kibana，就会在控制台上显示如下信息：

```
Loading the hack for branding plugin.
```

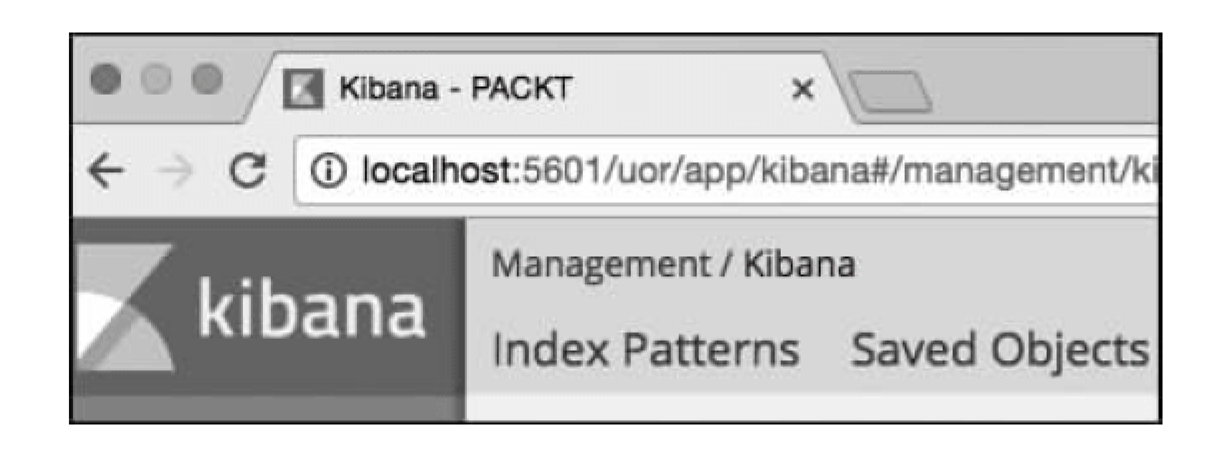
如果有类似这样的日志信息输出，就说明插件已经正常工作了，下面就可以着手完成其他功能了。

为其加标签 branding 时，还有很多针对 Kibana 需要完成的修改工作（例如变换 logo，修改页标题，修改颜色模式，定义页脚，设置 CSS 整体布局等）。总的来说，在浏览器中见到的任何部分都可以被加以修改、完善，以便满足实际需求。为便于理解，下面通过追加 PACKT，实现一个简单的修改每一页标题的小例子。为相关代码更新 hack.js 文件。最终的 hack.js 文件应该类似如下：

```
import $  from 'jquery';
```

```
    console.log("Loading the hack for branding plugin.");
    $ (document).ready(function(){
        document.title =document.title +" -PACKT";
});
```

上述代码将会修改载入的 Kibana 中每一页的标题 title。如果服务器已经运行,一旦修改了位于插件文件夹中的文件,服务器就会自动重新启动来更新插件。刷新浏览器时,就能看到页标题已经改变了。



从上面给出的截图中可以看出,这是一个非常简单的对页面标题进行修改的例子。其中使用了 jQuery(能提供强大的修改 DOM 的功能)。修改标题只是冰山一角,还可以完成很多事情。注意,使用 jQuery 并不是唯一的方式,也不是完成上述功能的极简方式。如果可能,则应该考虑使用 Node.js 模块及其元素来完成上述功能。

7.5 本章小结

如果讨论针对四个不同工具的个性化的做法,那么应该用四个不同的部分来分别说明。然而,这里只是希望能理解一个工具是如何被扩充其功能的。

本章中,通过学习如何组织一个插件并对其进行个性化的定制,学习了相应的扩充 Elasticsearch 功能的方法。随后,生成了一个 Logstash 插件,用于从 openweathermap.org 中捕获相应的数据,并将它们发送到 Elasticsearch 中。我们也学习了如何生成从同样的 Web 服务器中搜集数据的 Beat。最后,搭建了 Kibana 开发环境,学习了如何生成基础的、通过追加 PACKT 的方式修改 Kibana 页面标题的方法。

下一章中,将会再学习 Elasticsearch 及其相应的 API 的用法,将会学习如何获取 API 及其 API 的作用等,并学习在 Elasticsearch v5.x 中新增加的 Ingest 节点的方法。



Elasticsearch API

在第 2 章中，学习了 Elasticsearch 的基本技术，了解了 Elasticsearch 如何工作，以及它提供了哪些 API。在第 4 章中，了解了如何使用控制台(Console)，并且开始使用 Kibana 中的聚合。

本章将介绍其余的 API，将使用控制台向 Elasticsearch 发送 API 请求。本章介绍以下主题：

- 集群 API；
- Cat API；
- 模块；
- Ingest 节点；
- Elasticsearch 客户端；
- Java API。

要快速熟悉控制台，可以参考第 4 章中的探索开发者工具 Dev Tools 的内容。

> 如果目的是开发和学习，就可以在本地安装 Kibana。

8.1 集群 API

集群 API 允许我们了解集群的状态、健康状况、统计、节点统计数据、节点信息等。

8.1.1 集群健康状况

想要了解集群的健康状况，可以使用_cluster/health，如下所示：

```
GET /_cluster/health/library
```

在这里，GET 是请求的方式，_cluster/health 是参数，library 是索引名。执行此调用将返回相关信息，包括节点、数据节点、分片、任务和索引状态等。这里指定了 library 索引

名，这样就可以只获取 library 索引的有关信息：

Dev Tools　History　Settings　Help

Console

```
GET /_cluster/health/library
```

```
{
  "cluster_name": "elasticsearch",
  "status": "green",
  "timed_out": false,
  "number_of_nodes": 2,
  "number_of_data_nodes": 2,
  "active_primary_shards": 5,
  "active_shards": 10,
  "relocating_shards": 0,
  "initializing_shards": 0,
  "unassigned_shards": 0,
  "delayed_unassigned_shards": 0,
  "number_of_pending_tasks": 0,
  "number_of_in_flight_fetch": 0,
  "task_max_waiting_in_queue_millis": 0,
  "active_shards_percent_as_number": 100
}
```

响应窗口中的结果显示了 Elasticsearch 集群的健康状况为绿色，同时也显示了其他信息，例如节点信息、分片和待处理的任务。

让我们看看 Elasticsearch 状态的其他值意味着什么：

- **红色**：某些或所有主分片未被分配或未就绪。
- **黄色**：所有主分片都已被分配，但是一些或全部副本分片未被分配。
- **绿色**：所有主分片和副本分片都被分配，集群已经完全启动。

如果副分片没有定义，就可以在控制台中使用这样的请求来定义它们：

```
PUT /_settings
    {   "index": {
        "number_of_replicas": 1
    }
}
```

下表是其他请求参数及其描述。

请求参数	描述
level	要返回的集群健康信息的级别，可选的值有：集群(默认值)、索引或分片
wait_for_status	可能的值有红色、黄色、绿色，集群将会等待其健康状况变为指定的值
wait_for_relocating_shards	要等待多少分片被迁移？默认不等待，值为 0 表示等到所有分片迁移为止
wait_for_active_shards	要等待多少分片成为活跃分片？默认不等待

续表

请求参数	描　述
wait_for_nodes	等待指定数量的节点变为可用
timeout	如果提供了 wait_for_xxx，则在这里指定等待的秒数
local	默认情况下，信息由主节点提供；如果设置 local＝true，则提供本地节点的信息

要使用这些请求参数，只需添加一个查询参数并提供其值。例如，如果想要获得索引级别的集群健康状况，可以执行以下命令。

```
GET _cluster/health?level=indices
```

执行此命令将返回集群中索引的详细信息。例如 library 索引的信息如下所示：

```
"indices": {
        "library": {
        "status": "green",
        "number_of_shards": 5,
        "number_of_replicas": 1,
        "active_primary_shards": 5,
        "active_shards": 10,
        "relocating_shards": 0,
        "initializing_shards": 0,
        "unassigned_shards": 0
      }
}
```

相似地，要使用 wait_for_status，可以执行以下命令：

```
GET /_cluster/health?wait_for_status=yellow
```

还可以使用多个参数，例如：

```
GET /_cluster/health?wait_for_status=yellow&level=indices
```

在等待状态为黄色时查看索引级别的信息。

8.1.2　集群状态

这个 API 提供了关于集群状态非常详细的信息。若要查看完整状态，可以执行以下命令：

```
GET /_cluster/state?pretty
```

如果想将请求范围控制在指标 metric 和索引中，可以执行以下命令：

```
GET /_cluster/state/{metrics}/{indices}?pretty'
```

指标和索引都可以包含多个由逗号分隔的条目。指标可以是版本、主节点 master_node、节点、路由表 routing_table、元数据、数据块 blocks、_all(用于全部指标)。

> 控制台会自动以良好的缩进格式打印出结果。因此，当使用控制台运行时，不需要在 API 请求的结尾处添加?pretty。

8.1.3 集群统计信息

这个 API 提供了有关集群中索引和节点中基本指标的信息，包括分片数、节点数、内存使用、角色、JVM 信息、CPU 使用情况、插件信息等。可以使用下面的 URI 来执行：

```
GET /_cluster/stats
```

8.1.4 待处理任务

此 API 将返回群集中所有待处理的任务。如果没有待处理任务，它将返回一个空的任务数组：

```
GET /_cluster/pending_tasks
```

8.1.5 集群重路由

这个 API 更像是用于执行任务的集群的命令。这里有三种可用的命令：move、allocate、cancel，使用方法是_cluster/reroute。

下面的表格列出了每个命令的详细信息：

命　　令	功　　能
move	将分片从一个节点移动到集群的其他节点
allocate	将一个未分配的分片分配到某个节点
cancel	取消一个分片的分配

让我们看看下面这个例子，它将已启动的分片从一个节点移动到另一个节点：

```
POST _cluster/reroute
    {
        "commands": [
```

```
            {
                "move": {
                    "index": "library",
                    "shard": 2,
                    "from_node": "node_1",
                    "to_node": "node_2"
                }
            }
        ]
    }
```

执行此命令后，将从 node_1 移动 library 索引的其中 2 个分片到 node_2 中。

8.1.6 集群更新设置

Elasticsearch 的配置信息存储在 config 目录下的 elasticsearch.yml 配置文件中。这些配置大多是静态的。这里也包含集群的动态配置，可以由相应的 API 来动态配置。这些设置可分为两类，一种是持久的，即使在一个集群完全重启后，它也是适用的；另一种则是临时的，一旦集群重新启动，配置将会丢失。临时设置优先于持久性设置，持久性设置优于 elasticsearch.yml 配置文件中设置。推荐对本地节点使用 elasticsearch.yml 配置文件来配置，对集群级别的节点使用 API 来设置(而不推荐使用此 endpoint 参数来更新设置)。更多关于设置的文档可在以下网址中找到：https://www.elastic.co/guide/en/elasticsearch/reference/5.1/settings.html

为获得所有这样的设置，可以使用_cluster/settings 参数：

```
GET /_cluster/settings
```

要设置这些配置，可以使用相同的参数：

```
PUT /_cluster/settings
{
    "persistent": {
        "discovery.zen.minimum_master_nodes": 2
    }
}
```

8.1.7 节点统计信息

与集群统计信息类似，可以获取一个或多个节点的统计信息：

```
GET /_nodes/stats
```

在默认情况下，程序会返回所有节点的数据；可以指定节点 ID 和以逗号分隔的其他值，例如 indices、os、process、jvm、transport、http、fs、thread_pool、breaker 等。

要获得具体信息，可以执行以下命令：

```
GET /_nodes/stats/fs,indices
GET /_nodes/node1/stats/fs,indices
```

第一个命令将获得文件系统和各节点的索引信息，第二个命令中的 URI 只会得到 node1 的文件系统和索引。

8.1.8 节点信息 API

这个 API 可以获取节点信息。为了获得一个或所有节点的进程信息，可以执行以下命令：

```
GET /_nodes/os,process
GET /_nodes/_all/os,process
GET /_nodes/node1/os,process
```

第一个命令和第二个命令是相同的，它们将获得所有节点的信息。上述命令中的 uri 可以获取操作系统和进程相关的信息。我们可以得到的其他信息有 settings、jvm、transport、http、thread_pool、plugins。执行上述命令将返回以下内容：

```
{
    "_nodes": {
      "total": 1,
      "successful": 1,
      "failed": 0
    },
    "cluster_name": "elasticsearch",
    "nodes": {
      "U4MPVritSmmAyoHLb2dLzw": {
        "name": "node1",
        "transport_address": "127.0.0.1:9300",
        "host": "127.0.0.1",
        "ip": "127.0.0.1",
        "version": "5.1.1",
        "build_hash": "5395e21",
        "roles": [
          "master",
          "data",
          "ingest"
```

```
          ],
          "os": {
            "refresh_interval_in_millis": 1000,
            "name": "Mac OS X",
            "arch": "x86_64",
            "version": "10.0",
            "available_processors": 4,
            "allocated_processors": 4
          },
          "process": {
            "refresh_interval_in_millis": 1000,
            "id": 15188,
            "mlockall": false
        }
      }
    }
}
```

要在集群中的节点上获取 hot threads，可以使用_nodes/hot_threads 或者 _nodes/nodeId/hot_threads。

```
GET /_nodes/hot_threads
```

8.1.9 任务管理 API

这个 API 帮助我们获取集群中的所有任务，或者一个特定的节点的任务。它也允许我们执行一些操作。要获取集群中所有任务，可以执行以下命令：

```
GET /_tasks
```

执行后，会返回以下输出信息：

```
{
"nodes" : {
      "U4MPVritSmmAyoHLb2dLzw" : {
        "name" : "node1",
        "transport_address" : "127.0.0.1:9300",
        "host" : "127.0.0.1",
        "ip" : "127.0.0.1:9300",
        "roles": [
          "master",
          "data",
```

```
            "ingest"      ],
          "tasks" : {
            "U4MPVritSmmAyoHLb2dLzw:4129" : {
              "node" : "U4MPVritSmmAyoHLb2dLzw",
              "id" : 4129,
              "type" : "transport",
              "action" : "cluster:monitor/tasks/lists",
              "start_time_in_millis" : 1463037154374,
              "running_time_in_nanos" : 1205000,
              "cancellable": false,
            },
            "U4MPVritSmmAyoHLb2dLzw:4130" : {
              "node" : "U4MPVritSmmAyoHLb2dLzw",
              "id" : 4130,
              "type" : "direct",
              "action" : "cluster:monitor/tasks/lists[n]",
              "start_time_in_millis" : 1463037154344,
              "running_time_in_nanos" : 30881000,
              "cancellable": false,
              "parent_task_id": " U4MPVritSmmAyoHLb2dLzw:4129"
          }
        }
      }
    }
  }
```

正如所看到的,集群中列出了两个任务,也可以从指定的节点中获取类似的信息:

```
GET /_tasks?nodes=node1
```

甚至可进一步使用 actions 过滤结果,例如:

```
GET /_tasks?nodes=node1&actions=cluster:
```

正如所看到的,集群中列出了两个任务,也可以从指定的节点中获取类似的信息。

8.2 Cat API

这个 API 以可读的格式(而不是原始的 JSON 格式)显示节点、索引、字段、任务和插件的详细信息。利用 Console,还可以在控制台上看到程序是如何为表格的可视化内容提供数据的。

所有这些命令都可以用 curl 以 GET 方式执行。默认情况下，执行这样的命令只会列出数据，而没有表头。为了显示表头，可以使用查询参数 v：

```
GET /_cat/health?v
```

可以使用上面这条命令而不用以下命令：

```
GET /_cat/health
```

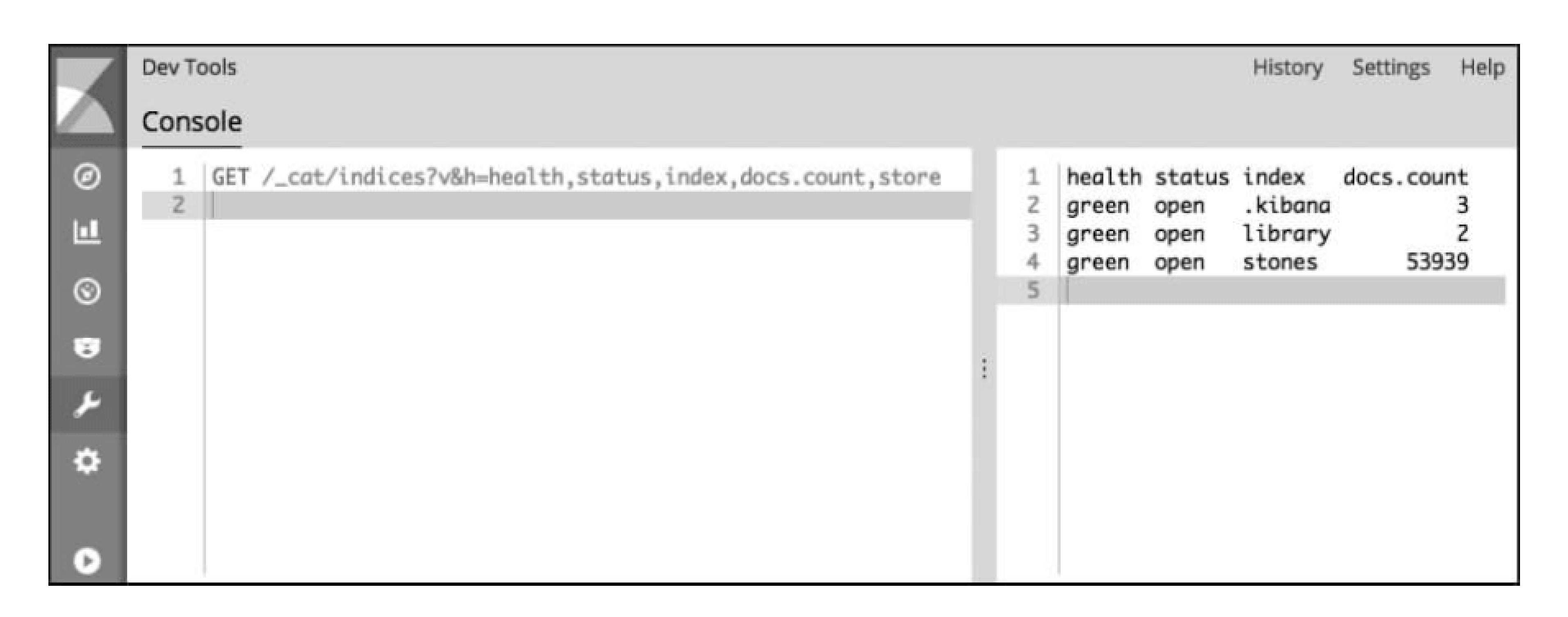
还可以在查询参数 h 中提供以逗号分隔的值来指定要显示哪些字段的表头。

让我们看看可用于操作的参数选项：

- _cat/indices：该选项显示了诸如健康状况、索引状态、索引名称、主分片、副分片、文档计数和内存使用情况等索引数据：

  ```
  GET /_cat/indices?v&h=health,status,index,docs.count,store
  ```

下图是控制台显示的结果：



正如我们所看到的，它显示了每个索引的健康状况和统计信息。

- _cat/master：该选项显示了主节点的节点 ID、IP 地址和主节点名称，如下所示：

  ```
  GET /_cat/master?v
  ```

  执行这一命令将返回下面的输出：

  ```
  id                     host      ip        node
  U4MPVritSmmAyoHLb2dLzw 127.0.0.1 127.0.0.1 node1
  ```

- _cat/nodes：该选项展示了集群中的节点信息，其中有许多可以使用的表头，比较重要的有 id、pid、name、master(节点是否为主节点)、主机、ip、port、version(ES 的版本)、Build、jdk，以及 disk、heap、ram 和文件描述符信息：

  ```
  GET _cat/nodes?v
  ```

  结果如下：

```
ip        heap.percent ram.percent cpu load_1m load_5m load_15m node.role
master name
127.0.0.1 13 47 -1                          mdi * node1
127.0.0.1 13 47 -1                          mdi -node2
```

- _cat/health：该选项显示了一个时间戳位置的集群健康状况，也显示了全部的节点、分片和任务的健康状况：

  GET /_cat/health

  结果如下：

  ```
  1482738453 13:17:33 elasticsearch green 2 2 12 6 0 0 0 0 -100.0%
  ```

  可以指定一些表头，方式如下所示：

  GET/_cat/health?v&h=cluster,status,node.total,shards,pending_tasks

  结果如下：

  ```
  cluster       status node.total shards
  pending_tasks elasticsearch green 2     12            0
  ```

- _cat/allocation：该选项列出了所有给节点、磁盘空间(消耗)等的分片分配信息：

  GET _cat/allocation?v

- _cat/shards：这个选项可以详细展示每个节点上的分片信息。对于每一个分片，其中显示了其索引、类型、状态、文档数量、内存使用情况、所在节点等信息：

  GET _cat/shards?v

执行该命令，将输出每个索引中的分片详细信息。在下面的列表中，显示了一些library索引的信息：

Index	shard	prirep	state	docs	store	ip	node
library	2	r	STARTED	0	159b	127.0.0.1	node1
library	2	p	STARTED	0	159b	127.0.0.1	node2
library	4	r	STARTED	1	3.3kb	127.0.0.1	node1
library	4	p	STARTED	1	3.3kb	127.0.0.1	node2
library	3	r	STARTED	2	9.4kb	127.0.0.1	node1
library	3	p	STARTED	2	9.4kb	127.0.0.1	node2

- _cat/aliases：该选项显示索引中所有配置的别名：

  GET _cat/aliases

要得到一个特定的别名，可以使用_cat/aliases/{aliasname}。

> ℹ如果没有定义任何别名，则在相应窗格中得到一个空白的输出。

- _cat/plugins：该选项可以列出每个节点上已安装的所有插件的名称、版本、插件类型、插件的 URL 等信息：

```
GET _cat/plugins
```

> ℹ如果没有使用任何插件，则在相应窗格中得到一个空白的输出。

- _cat/count：该选项提供了一个索引或整个集群中(默认)的文档计数。为了得到索引的文档计数，可以使用_cat/count/{indexname}，仅显示一个实时文件计数。
- _cat/fielddata：该选项通过 fielddata 列出每个数据节点上的堆内存使用情况。我们还可以使用字段查询参数来指定特定字段。
- _cat / nodeattrs：如果自定义节点属性存在，该选项可以将其显示出来。
- _cat / pending_tasks：该选项将列出集群中的所有待处理任务。此列表显示了 insertOrder、timeInQueue、priority、source 等数据。
- _cat/recovery：该选项显示了索引分片做的所有恢复操作和正在进行的工作。对于每一个恢复操作，会显示索引、分片 ID、时间、类型、所处的阶段(例如已完成)、源和目标主机、总文件和字节数、快照和存储信息等数据。
- _cat/repositories：该选项可以列出所有存储库的 ID 和类型。
- _cat/thread_pool：该选项可以显示每个节点的集群级线程池。对于每个节点，可以显示其中活跃(active)、排队(queued)、拒绝(rejected)等不同的线程。可用的线程池有 bulk、fetch_shard_started、fetch_share_store、flush、force_merge、generic、get、index、listener、management、 refresh、search、snapshot、warmer 等。
- _cat/segments：该选项显示分片中各个分段的底层信息。
- _cat/snapshots/{repository}：如果存在一个存储库，该参数将显示它的所有快照，其中{repository}指定了存储库的名称。

8.3 Elasticsearch 模块

每一个伟大的项目，其功能都由若干个模块组成并由这些模块支撑，Elasticsearch 中有很多这样的模块。这些模块需要一些设置(无论是动态的设置，或者使用 elasticsearch. yml 文件完成的静态设置)，都可以使用群集 API 来更新。让我们来看看不同的模块。

8.3.1 集群模块

为了保持集群的负载均衡，集群模块负责如何将分片分配给节点并处理这些分片在集群内的移动。这个过程被称为分片分配。此模块有许多设置，可以使用集群 API 动态地应用这些设置。这些设置包括集群中节点分配的设置，以及节点内部的设置。

8.3.2 Discovery 模块

Discovery 模块有助于发现网络中指定集群的节点。在 elasticsearch.yml 配置文件中有一个集群名称的配置项，它决定了当前节点是哪个集群的一部分。该项默认名称是 elasticsearch：

```
cluster.name =elasticsearch
```

所有配置了同一集群名称的节点将归为同一个集群，该模块还决定了哪个节点将作为主节点。

8.3.3 Gateway 模块

有些时候我们的集群需要恢复，在这种情况下，对于有多少节点应该在恢复开始之前启动等问题，都是在配置信息中定义的。这些设置是静态的，必须对群集中的每个节点进行配置。

8.3.4 HTTP 模块

所有的 Elasticsearch API 都由这个模块在 HTTP 上异步地开放，所有的设置都是静态的，是在 elasticsearch.yml 配置文件中定义的。如果不想调用 REST 命令，也可以禁用此模块。集群中的节点使用 Transport 模块进行通信，将 http.enabled 设置为 false，可以禁用此模块。

8.3.5 索引模块

所有与索引相关的设置(适用于所有索引)都由该模块来管理。这里有很多设置可以控制异常、缓存、索引缓冲区、请求缓存等。

8.3.6 网络模块

该模块负责网络相关的设置，并且帮助我们建立生产级别的节点和集群。默认情况下，所有设置都适用于本地主机(127.0.0.1)。在生产环境中，应该绑定适当的主机名和端口，许多相关的网络设置由该模块控制。

8.3.7 节点客户端

节点可以分为几种类型，分别如下：

- **主节点 master node**：这种节点可以控制集群，设置 node.master 属性为 true，来赋予主节点资格。集群启动后，将在这个类型的节点中选择主节点：

```
node.master : true
```

- **数据节点 Data node**：该节点可以执行 CRUD、搜索和聚合。要使当前节点仅为数据节点，只需将 node.data 属性设置为 true：

```
node.data : true
```

- **Ingest 节点**：这是在 5.x 版本之后加入的一种新的节点类型。默认情况下，所有节点都能作为 Ingest 节点，如果不希望某些节点成为 Ingest 节点，就需要禁用它们。在对文档执行任何操作之前，这些节点都会进行预处理。稍后将在本章详细讨论这些节点。
- **Tribe 节点**：这些节点可以连接到多个集群，并在集群中执行必要的操作。这些节点是一种特殊类型的用来协调的节点。

总之，对于每种类型的节点，请参阅下表中的 elasticsearch.yml 设置。

主　结　点	数 据 节 点	Ingest 节点	Tribe 节点
node.master: true node.data: false node.ingest: false	node.master: false node.data: true node.ingest: false	node.master: false node.data: false node.ingest: true	node.master: false node.data: false node.ingest: false

如果使用这些设置，该节点将只具备四种功能中的一种。

8.3.8 插件模块

插件扩展了 Elasticsearch 并提供了更多功能。在本章后面的部分，将对 Elasticsearch 插件进行介绍。

8.3.9 脚本

这个模块允许使用 API 的时候执行带有脚本的操作，例如脚本可以用来更新索引中的文档。Elasticsearch 支持多种脚本语言，其中 Groovy 和 Painless 是最流行的两种，这些插件都是内置的。使用脚本并不总是安全的，因此 Elasticsearch 有一套启用和禁用脚本的设

置。这些设置都是细粒度的，它们分为两种形式：

第一种形式如下：

```
script.engine.{lang}.{source}.{context}: true|false
```

第二种形式如下：

```
script.engine.{lang}.{inline|file|stored}: true|false
```

例如，要启用 inline Groovy 脚本，可使用如下方法：

```
script.engine.groovy.inline: true
```

要启用、禁用所有语言的脚本，可以使用下面示例中的属性，这里设置允许使用 inline 脚本：

```
script.inline: true
```

建议使用细粒度的底层（而不是这种高层次）的设置，因为这样可以让任何人在你的 Elasticsearch 集群上执行任何脚本。

建议在 Groovy 的基础上使用 Painless，这样可以增加安全性和性能优势。

8.3.10 快照/恢复模块

这个模块可向 Elasticsearch 中（以及从 Elasticsearch 向外）备份和恢复数据。该操作可以在整个集群中执行，也可在单个索引中执行。在创建快照之前，必须搭建一个存储库，这里也有官方提供的用于 AWS Cloud、HDFS、Azure cloud 存储库的插件。

8.3.11 线程池

高效的线程管理和内存管理是搜索引擎的一个关键因素。每个 Elasticsearch 节点都有一些用于不同用途的线程池，这些用途包括通用目的、索引、搜索、获取数据、bulk、percolate、快照、warmer、刷新，以及监听器操作。

8.3.12 Transport 模块

我们已经了解到 Elasticsearch 节点有 HTTP 层和 **Transport** 层。HTTP 层服务于 REST 请求，**Transport** 层用于集群和节点之间的内部通信。传输模块具有 TCP 的实现，其中有许多可用的端口、主机、超时等设置。这里还有一个专用的日志记录程序，用于记录所有传入和传出请求的日志。这个日志记录程序称为 **Transport Tracer**。

8.3.13 Tribe 节点模块

Tribe 节点是一个特殊的客户端节点，它可以跨多个集群进行通信。在其配置文件中，Tribe 节点应该拥有一个所有已加入集群的列表。同时，Tribe 节点应该能够连接列表所记载集群中的所有节点，并且能够在其上执行读写操作。

8.4 Ingest 节点

正如在前面章节中所学到的，在为文档创建索引之前，Ingest 节点能对其进行预处理。在执行批量请求或索引操作之前，Ingest 节点会监听请求，并对文档进行必要的处理。这种处理器的一个示例是日期处理器，它用于解析字段中的日期。另一个例子是转换处理器，它将字段值转换为目标类型，例如将字符串类型转换为整型。大量的此类处理器均可访问该网址获取：https://www.elastic.co/guide/en/elasticsearch/reference/5.1/ingest-processors.html。

当一个体量十分巨大的处理操作执行时，如果不希望数据节点或主节点参与这些处理，Ingest 节点是很有帮助的。专门用来执行处理操作的 Ingest 节点可以显著降低负荷。最好将数据节点和主节点的 node.ingest 属性设置为 false。

为了了解 Ingest 节点如何与数据管道协同工作，可以按照以下步骤来执行。以第 2 章中的 library 索引和增加的类型 movies 为例子：

(1) **决定要处理什么**：在数据集当中，有 20 多个字段，现在又在其中添加了 7 个类型(genre)字段。这些新加入的字段是纪录片 documentary、动作片 action、戏剧 drama、短剧 short、动画 animation、浪漫剧 romance、喜剧 comedy。所有这些字段都采取二进制的取值，其中只有一个值代表 true。然而，这可能被合并成一个名为 genre 的单独的字段，以便于更好地理解、分析和可视化。

如果已经有 library 索引的 movies 类型，请删除此类型，以便在开始操作之前有一个全新的索引。可在 library 索引上调用_delete_by_query API 来删除 movies 类型：

```
POST library/_delete_by_query{ "query":{ "match": { "_type":"movies" } }}
```

(2) **准备处理器**：可以用 Painless 脚本检查是否有任何字段的值设置为 1，然后添加一个新的字段 genre，并将值为 1 的字段名设置为其值。下面的脚本可以做到相同的操作：

```
{
    "script": {
        "inline": " if(ctx.romance ==1){ ctx.genre ="romance"}
```

```
else if(ctx.drama ==1){ ctx.genre ="drama"} else
if(ctx.action ==1){
    ctx.genre ="action"} elseif(ctx.documentary ==1){ ctx.genre =
"documentary"}
    else if(ctx.comedy ==1){ ctx.genre ="comedy"} else
if(ctx.animation ==1){
    ctx.genre ="animation"} elseif(ctx.short ==1){ ctx.genre =
"short"} ",
    "lang": "painless"
}
}
```

执行这个脚本,将在文档中添加一个名为 genre 的新字段。

(3) **模拟管道**:模拟管道是检查问题最好的方式。要对管道进行模拟,可以使用 _ingest/pipeline/_simulate参数。管道模拟器需要两个字段——pipeline(包含要测试的管道)和 docs(要进行测试的示例文档)。调用的模拟功能及其管道的相关命令如下所示:

```
POST _ingest/pipeline/_simulate
{
    "pipeline": {
        "description": "Process Genre Data",
        "processors": [
            {"script": {
        "inline": "if(ctx.romance ==1){  ctx.genre ="romance"}
    else  if(ctx.drama ==1){ ctx.genre ="drama"}
    else   if(ctx.action ==1){ ctx.genre ="action"}
    else  if(ctx.documentary ==1){ ctx.genre ="documentary"}
    else  if(ctx.comedy ==1){ ctx.genre ="comedy"}
    else if(ctx.animation ==1){ ctx.genre ="animation"}
    else if(ctx.short ==1){ ctx.genre ="short"}",
        "lang": "painless"
            }
        }
    ]
}, "docs": [
        {
          "_index": "library",
          "_type": "movies",
          "_id": "1",
          "_source": {
            "romance": 0,
```

```
                "r2votes": 4,
                "year": 2001,
                "rating": 8,
                "r8votes": 14,
                "title": "Lord of the Rings: The Fellowship of the Ring ",
                "r5votes": 4,
                "drama": 0,
                "@version": "1",
                "action": 1,
                  "r6votes": 4,
                  "documentary": 0,
                  "budget": 93000000,
                  "r9votes": 14,
                  "comedy": 0,
                  "length": "208",
                  "r3votes": 4,
                  "r10votes": 45,
                  "tags": [],
                  "animation": 0,
                  "r4votes": 4,
                  "r7votes": 4,
                  "short": 0,
                  "votes": 157608,
                  "r1votes": 4,
                  "mpaaRating": "PG-13"
              }
          }
      ]
  }
```

如果看到最后以此调用的输出,就应该能够看到一个添加到文档的新字段 genre。

(4) **插入管道**:管道包含两个字段:description 和 processors,这里可以定义多个处理器。可以使用_ingest/pipeline 参数来插入管道,并为其设置一个名字,这里将其名字设置为 process_genre_data。调用 API 来插入管道的命令如下所示:

```
PUT _ingest/pipeline/process-genre-data
{
    "description": "Process Genre Data",
    "processors": [    {"script": {    "inline":
    "if(ctx.romance ==1){  ctx.genre =     "romance"}
    else if(ctx.drama ==1){ ctx.genre ="drama"}
```

```
        else if(ctx.action ==1){ ctx.genre ="action"}
        else if(ctx.documentary ==1){ ctx.genre ="documentary"}
        else if(ctx.comedy ==1){ ctx.genre ="comedy"}
        else if(ctx.animation ==1){ ctx.genre ="animation"}
        else if(ctx.short ==1){ ctx.genre ="short"}",
            "lang": "painless"
              }
          }
        ]
}
```

如果运行该命令,控制台将输出一个肯定的应答。

(5) **使用管道**:管道设置好之后,即可使用它。在第 2 章中,用 Logstash. movies. conf 配置文件为电影数据创建索引。除了一处 Elasticsearch 输出配置上的修改之外,其余部分将保持相同的配置。需要设置管道的字段,以便 Elasticsearch 能够使用该管道。更新后的输出配置如下:

```
output {
    elasticsearch {
        index =>"library"
        document_type =>"movies"
        hosts =>"localhost"
        pipeline =>"process-genre-data"
    }
}
```

配置更新后,就可执行索引过程。假设(仅)用 Ingest 节点建立了一个集群,下面运行 Logstash:

```
./bin/Logstash -f conf/Logstash.movies.conf
```

(6) **验证 Ingest 过程**:通过查询电影数据,验证管道工作状态是否良好,应该能够看到一个新添加的字段 genre。要在 Kibana 控制台测试这个过程,可以运行下面的命令:

```
GET library/movies/_search{  "query": {    "match_all": {    }  }}
```

在命中的搜索结果中,你应该看到每个电影的文档都增加了 genre 字段。

这是一个简单的关于管道的例子,其中使用了 Ingest 节点和 Logstash。Ingest 节点是 Elastic Stack 的一个新成员,实现管道定义工作,可以在不使用 Logstash 的情况下直接使用。Ingest 节点在功能、处理器等方面还在发展演进,目前已经比 Logstash 表现出更高的性能和处理文件的吞吐量。

在 Logstash 中使用的带有管道的输出配置方式，使用带有管道的 Beats 组件同样可以做到。在这种情况下，如果使用 Ingest 节点可以实现，就可以省去对 Logstash 的使用。以下是 Filebeat 组件使用管道的示例：

```
output.elasticsearch:
hosts: ["http://localhost:9200"]
pipeline: process_genre_data
```

在 Elasticsearch 的输出配置中，只需要指定管道的名称即可。

8.5 Elasticsearch 客户端

在本章前面的部分，我们了解了 Elasticsearch 节点支持 Transport 和 HTTP 协议。利用这种灵活性，Elasticsearch 节点可以接受客户端(client)的管理，这种客户端是由其他编程语言编写的。有很多客户端可以在集群和节点上执行操作。这些客户端可以连接到节点或群集上，管理索引，对文档操作，以及执行搜索。

8.5.1 支持的客户端

某些客户端由 Elasticsearch 官方支持。这些客户端基本上都是 API，可以用熟悉的语言编写应用程序来调用它们。例如，如果正在开发 JavaWeb 应用程序，要集成到 Elasticsearch，同时想提供管理员面板来管理索引，就可以调用 Elasticsearch 支持的 JavaAPI 连接到群集和节点，完成必要的操作。以下是所有 Elasticsearch 支持的客户端列表：

- Java API；
- JavaScript API；
- Groovy API；
- .NET API；
- PHPAPI；
- Perl API；
- Python API；
- Ruby API。

8.5.2 社区提供的客户端

开源软件之所以强大，是因为它来源于社区。Elasticsearch 也是这样，社区的贡献在其中起到了重要的作用。Elasticsearch 官方仅正式支持少数客户端，但社区贡献了 30 多个。

这些客户包括其他语言例如 Go、Scala、R、Smalltalk 等。虽然不是所有的客户端都支持全部的操作,但至少可以支持最常见的以及有用的操作。

完整的客户端列表可访问以下网址来获取:https://www.elastic.co/guide/en/elasticsearch/client/community/current/index.html。

8.6 Java API

Java 客户端使用传输层来执行操作,它可以支持各种操作。借助它可以搜索,为文档创建索引,删除或者获取包括集群管理任务在内的文档;也可以批量执行操作。

要在应用程序中使用 JavaAPI,需要使用 JAR 文件作为依赖。对于 Maven 项目,可以用 pom.xml 添加如下所示的依赖:

```
<dependency>
    <groupId>org.elasticsearch</groupId>
    <artifactId>elasticsearch</artifactId>
    <version>$ {elasticsearch.version}</version>
</dependency>
```

为了将 JAR 文件直接引入到这个项目中,可以从远端存储库 https://repo.maven.apache.org/maven2/org/elasticsearch/elasticsearch 中下载,可以选择适合的版本。

需要注意的是,客户端版本应该与当前使用的 Elasticsearch 版本相同。例如,如果使用 Elasticsearch-5.1.1,那么也应使用 5.1.1 版本的相应的 JAR 文件。对于 pom.xml 文件,需要将变量 ${elasticsearch.version}的值设置为与正在使用 Elasticsearch 的版本相对应。

8.6.1 连接到集群

要连接到集群,可以利用 TransportClient 来创建客户端,如以下代码所示:

```
Clientclient=TransportClient.builder().build().addTransportAddress(
new InetSocketTransportAddress(InetAddress.getByName("localhost"), 9300));
```

节点启动后,注意 Elasticsearch 输出的日志,将打印出下面的代码:

```
[2016-05-02 16:39:51,832][INFO ][transport]
[Clive]publish_address{127.0.0.1:9300},
bound_addresses {127.0.0.1:9300},
{[::1]:9300}, {[fe80::1]:9300}
```

```
[2016-05-02 16:39:51,832][INFO ][http] [Clive] publish_address
{127.0.0.1:9200}, bound_addresses {127.0.0.1:9200}, {[::1]:9200},
{[fe80::1]:9200}
```

我们已经知道，Elasticsearch 支持传输层和 HTTP 层。Elasticsearch 需要不同的端口来连接这些层——默认情况下端口 9300 用于传输（Transport）层，端口 9200 用于 HTTP 层。

由于 Java 客户端运行在传输层，所以对 9300 端口进行连接。一旦连接客户端，就可以完成所需的操作。

一旦所有操作都完成，就应该调用 close()方法来终止连接：

```
client.close();
```

8.6.2 管理任务

为了使用 Java API 执行管理任务（admin task），需要使用 AdminClient，可以调用 client.admin()方法来获取它：

```
AdminClient admin =client.admin();
```

这个调用返回 Admin 静态类的一个对象，它实现了 AdminClient 接口。这个对象可对索引和集群进行操作。

8.6.2.1 管理索引

admin 对象使用自身提供的 admin.indices()方法提供 IndicesAdminClient 类的实例，可创建索引，刷新、获取设置，以及更新设置：

```
IndicesAdminClient indicesAdmin =admin.indices();
```

1. 创建索引

以第 2 章中使用的电影数据集为例。要创建带有默认设置的索引，可以编写以下代码实现：

```
indicesAdmin.prepareCreate("movies").get();
```

让我们来了解一下刚刚发生了什么——首先调用 prepareCreate()，并传入索引名称 movies 来准备索引；然后调用 get()方法来执行创建操作。调用 prepareCreate()方法将返回 CreateIndexRequestBuilder 类的对象，这是一个用于创建索引的 builder，要创建索引的时候可以调用它。

下面是一个更详尽的调用代码：

```
CreateIndexRequestBuilder indexBuilder =
indicesAdmin.prepareCreate("movies");
CreateIndexResponse createIndexResponse =indexBuilder.get();
```

这段程序将在集群上以默认设置创建一个索引，分片数为5，每个分片的副本数为1。要在创建索引时候指定不同的设置，可以调用 Settings 类中的 builder()方法：

```
indexRequestBuilder.setSettings(Settings.builder()
                    .put("index.number_of_shards", 2)
                    .put("index.number_of_replicas", 2));
```

可以继续添加重载的所有设置。setSettings()方法有多种形式，可以接收不同类型的输入(Settings. Builder 就是其中一个)。也可以向这个方法中传入一个 JSON 字符串。

2. 获得索引设置

在 GetSettingsResponse 类的实例中，可以调用 prepareGetSettings()方法准备设置信息，接着调用 get()来获取索引设置：

```
GetSettingsResponse settingsResponse =
indicesAdmin.prepareGetSettings("movies").get();
```

由于 preparegetsettings()方法可以接收任意数量的参数(varargs)，因此也可以一次获取多个索引的设置，这个方法的声明如下所示：

```
GetSettingsRequestBuilder prepareGetSettings(String... indices);
```

> ℹ 上述代码中的三个点表示该位置可以接收任意数量的参数，这也被称为 varargs (可变参数)。在参数数量未知的情况下，这种定义方式是很有帮助的。

可以调用 settingsResponse. getIndexToSettings()方法来迭代这个设置：

```
for (ObjectObjectCursor<String, Settings> ooCursor :
settingsResponse.getIndexToSettings()) {
    String index =ooCursor.key;
    Settings settings =ooCursor.value;
    Integer shards =settings.getAsInt("index.number_of_shards", null);
}
```

使用 cursor 对象，可以得到索引和设置。稍后，即可从 Settings 的对象中获取值。可以获取的其他 Settings 对象的值，如下所示：

```
index.creation_date
index.number_of_replicas
index.number_of_shards
index.uuid
index.version.created
```

3. 更新索引设置

要更新设置，可调用 prepareUpdateSettings()方法：

```
indicesAdmin.prepareUpdateSettings("movies").setSettings(Settings.builder().
put("index.number_of_replicas", 4)).get();
```

在上述语句中调用了 setSettings()方法，并指定 index. number_of_replicas 关键词更新设置，然后调用 get()方法执行操作。

4. 刷新索引

同样，可以调用 prepareRefresh()方法刷新一个或多个索引：

```
indicesAdmin.prepareRefresh("movies").get();
```

如果不指定任何索引名称，这个调用将刷新全部索引。

8.6.2.2 管理集群

IndicesAdminClient 提供了索引相关的任务，与之相似的是，Java API 提供了 ClusterAdminClient 管理集群相关的任务：

```
ClusterAdminClient clusterAdmin =admin.cluster();
```

可以初始化 ClusterAdminClient 的对象来执行集群相关的操作，例如管理任务、快照，了解集群和节点的健康状况，更新设置以及更多的工作。下面介绍一些这样的操作，它们都可以通过调用 Java API 来执行。

1. 获取集群任务信息

调用 ClusterAdmin 类中的方法准备一个包含所有任务的列表，返回 ListTasksRequest-Builder 的对象，可以使用它获取任务的响应信息。该响应包含所有 TaskInfo 对象的列表，可以对它进行迭代来获取任务信息。下面的代码是对所讨论的这部分内容的实现：

```
ListTasksRequestBuilder taskBuilder =clusterAdmin.prepareListTasks();
ListTasksResponse taskResponse =taskBuilder.get();
List<TaskInfo>tasks =taskResponse.getTasks();
for (TaskInfo taskInfo : tasks) {
```

```
    // get task information taskInfo.getId(),taskInfo.getDescription()
}
```

一旦获取响应信息，就可以得到所有任务的列表，并且通过迭代获取任务相关的信息。

2. 获取集群健康状况

通过 ClusterIndexHealth 类的对象可获取每个索引在集群中的健康状况信息：

```
ClusterHealthRequestBuilder healthBuilder =clusterAdmin.prepareHealth();
ClusterHealthResponse healthResponse =healthBuilder.get();
for (ClusterIndexHealth health : healthResponse.getIndices().values()) {
    String index =health.getIndex();
    int shardsCount =health.getNumberOfShards();
    int replicasCount =health.getNumberOfReplicas();
    ClusterHealthStatus status =health.getStatus();
}
```

得到了 **health** 对象之后，也可以获得其他信息，例如分片数量、副本和索引状态等。health 对象提供索引相关的信息时，healthResponse 对象可以提供集群级信息，例如节点数量、数据节点数量、群集名称以及活跃的分片等。

8.6.3 索引级任务

Java API 允许执行偏底层的任务，可以调用文档、搜索、计数以及其他可用的 API 来实现。Java API 可以同时对单个和多个文档执行操作，还可以使用聚合或查询 DSL 来构建查询，然后调用搜索 API 执行搜索。

8.6.3.1 管理文档

索引 API 可以向索引添加文档、更新文档和删除文档。

1. 索引文档

我们已经准备了一个客户端，下面将直接使用客户端来执行操作。

首先，编写一个创建索引的请求，其中包括一个特定的索引名称、一个类型名称，以及指定的一个 ID 号(可选)。这样，如果想将一个文档添加到 library 索引中，类型为 book，id 为 121；就可以编写以下代码来实现：

```
IndexRequestBuilder indexRequestBuilder =client.prepareIndex("library",
"book", "121");
IndexResponse response =indexRequestBuilder.setSource(<document>).get();
```

一旦构建了索引请求，就可以调用 setSource()方法，之后调用 get()方法执行操作。方法 setSource()有许多可能的调用方式，如下所示：

```
setSource(byte[] source)
setSource(byte[] source, int offset, int length)
setSource(BytesReference source)
```

这个调用以字节或 BytesReference 的形式将文档存入索引：

```
setSource(Map<String,?>source)
setSource(Map<String,?>source, XContentType contentType)
```

这种形式以 Map 的形式存储要用到的键值对(key-value-pair)，其中键(key)为字符串类型，值(value)为对象类型。这种形式的一个例子如下所示：

```
Map<String, Object>document =new HashMap<String, Object>();
document.put("author","Ravi");
document.put("title","Test-Driven JavaScript Development");
document.put("pages",240);
```

创建一个 Map 对象，并将作者、标题和页数的信息存入其中。

如果在准备文档时使用 JSON 字符串格式的数据，就可以直接利用这样的字符串来索引文档：

```
setSource(String source)
```

例如：

```
String book ="{" +""author":"Ravi"," +
    ""title":"Test-Driven JavaScript Development"," +
        ""pages":"240"" +
    "}";
IndexRequestBuilder indexRequestBuilder =client.prepareIndex("library",
"book");
indexRequestBuilder.setSource(book).get();
```

上述示例中没有提供 ID。在这种情况下，它将为文档自动生成一个 ID。

也可以用 XContentBuilder 创建索引文档的数据源：

```
setSource(XContentBuilder sourceBuilder)
```

setsource()方法还有一些其他的形式。它以键值对作为单独的参数，每次调用该方法时，最多可以传入四对键值对：

```
setSource(String field1, Object value1)
setSource(String field1, Object value1, String field2, Object value2)
setSource(String field1, Object value1, String field2, Object value2,
String field3, Object value3)
setSource(String field1, Object value1, String field2, Object value2,
String field3, Object value3, String field4, Object value4)
```

无论一个文档是否被创建，调用 get()均会返回一个响应，从这个响应中可以获取索引、类型、ID、版本和一个表明文档是否已被创建的布尔值：

```
response.getIndex()        // index name
response.getType()         // type name
response.getVersion()      // version number
response.getId()           // id of the indexed document
response.isCreated()       // true if the document was created
```

2. 获取文档

调用 prepareGet()方法可获取文档：

```
GetResponse getResponse =client.prepareGet("library", "book","121").get();
```

要获取文档，需要指定索引名称、类型名称和 ID 号。

3. 删除文档

调用 prepareDelete()[1]方法可删除文档：

```
DeleteResponse deleteResponse =client.prepareDelete("library", "book",
    "151").get();
```

要删除文档，需要指定索引名称、类型名称和 ID 号。

4. 更新文档

要更新文档，可以初始化 UpdateRequest 的对象，并指定索引的名称、类型、文档 ID，以及一个包含所要更新字段的文档对象：

```
UpdateRequest updateRequest =newUpdateRequest();
updateRequest.index("library");
updateRequest.type("book");
updateRequest.id("151");
updateRequest.doc(XContentFactory.jsonBuilder()
```

① 译者注：原文中这个方法名为 prepareGet()，笔者认为这里存在错误，将其改为 prepareDelete()。

```
        .startObject()
        .field("pages", "250")
        .endObject());
```

一旦准备好请求对象，即可调用 update()方法，之后调用 get()方法来执行最终的操作：

```
client.update(updateRequest).get();
```

另一种方法是调用 prepareUpdate()方法，然后调用 setScript()或 setDoc()方法：

```
updateRequest =new UpdateRequest("library", "book", "151")
    .script(new Script("ctx._source.pages ="250""));
client.update(updateRequest).get();
client.prepareUpdate("library", "book", "151")
    .setDoc(XContentFactory.jsonBuilder()
    .startObject().field("pages", "250").endObject()).get();
```

这里也支持 upsert。如果要更新的文档不存在，程序将创建一个新的文档。如果文档已预先创建好，程序将对它进行更新。调用 update()方法之前需要调用 upsert()方法来使用 upsert：

```
updateRequest.doc(XContentFactory.jsonBuilder().startObject().field("pages",
"250").endObject()).upsert();
    client.update(updateRequest).get();
```

8.6.3.2 查询 DSL 和搜索 API

这个 API 可以帮助我们使用 QueryBuilder factory 类的查询 DSL。使用这个 API 可创建简单和复杂的查询，然后使用搜索 API 在索引中执行搜索。

例如，要执行 matchAll 查询，可以编写如下代码来实现：

```
QueryBuilder queryBuilder =QueryBuilders.matchAllQuery();
```

用 QueryBuilder factory 类建立 matchAllQuery 并将其传入 QueryBuilder 对象。与之相似，也可以执行其他类别的查询：

```
QueryBuilders.matchQuery(String name, Object text)
```

这个方法执行后将返回 match 查询，其中带有两个参数——字段名称和要匹配的文本：

```
QueryBuilders.multiMatchQuery(Object text, String... fieldNames)
```

multiMatchQuery()的参数中，第一个参数指定要匹配的文本；第二个参数是可变参数 varargs，指定查询时要匹配内容的所有字段。

查询 DSL 中支持的查询类型非常多，其中包括连接查询、词项级检索、全文索引、地理信息检索、跨度查询等。完整的查询列表详见 https://www.elastic.co/guide/en/elasticsearch/client/java-api/5.1/java-query-dsl.html。

创建查询之后，即可使用搜索 API 来执行搜索操作。可在存有所有电影数据的 library 索引中执行搜索：

```
SearchResponse response =client.prepareSearch("library")
.setTypes("movies")      .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
.setQuery(QueryBuilders.matchQuery("title","titanic"))
.setFrom(0).setSize(60).setExplain(true)
.execute()
.actionGet();
```

上面的程序中调用了 prepareSearch()方法并提供了索引名称 library，然后调用 setTypes()方法并指定 movies 类型，紧接着调用 setSearchType()并指定搜索的执行方式。根据官方文档的描述，DFS_QUERY_THEN_FETCH 与 QUERY_THEN_FETCH 这两个名称除了开头部分 DFS 这一小段不同之外，其余部分都是相同的。带有 DFS 的方式在执行和计算分布式文件系统中的词频时，可以得到更准确的评分。

使用 QUERY_THEN_FETCH 时，程序将对所有分片执行查询，但文件的内容不会立刻被返回，只有关于结果的一部分必要的（最小的）的信息会被返回。程序根据查询来对结果进行整理和排序，并获取到文档的内容。获取文档内容时，只涉及与之相关的分片。

设置搜索类别之后，又设置了一个查询。这个查询将使用 QueryBuilders 创建的查询对象。然后指定从第几条查询至第几条的数量信息，以及是否想要返回关于文档如何被评分的解释信息。最后，对 execute()和 actionGet()方法的调用执行查询，并得到响应。这个响应信息包含了时间信息、分片信息以及查询中命中的文档信息。

8.6.3.3 聚合

Java API 提供了一种全新的方式，利用 AggregationBuilders 实用程序实现聚合。一个典型的搜索操作如下所示：

```
SearchResponse searchResponse =client.prepareSearch("stones")
.addAggregation(aggregation)
.execute().actionGet();
```

在其中的 execute()方法调用之前可以附加一个聚合。要在 library 索引的 book 类型中创建一个统计最小页数的聚合，可以编写如下代码来实现：

```
MetricsAggregationBuilder aggregation =
                        AggregationBuilders.min("min_pages").field("pages");
SearchResponse searchResponse = client.prepareSearch("library")
                             .addAggregation(aggregation)
                             .execute().actionGet();
Min min_pages_agg = searchResponse.getAggregations().get("min_pages");
long min_pages = (long)min_pages_agg.getValue();
```

在这段代码中，调用 min()方法创建了一个 pages 字段上的 metric 聚合。执行搜索操作之后，需要通过提供聚合名称从搜索响应信息中获取聚合结果，然后调用该聚合的方法，即可得到所需的值。

类似于 Min 聚合，指定其他聚合的类别就可以得到其他 metric 聚合。也可以执行 bucket 聚合，然后从 searchResponse 中提取结果。

8.7 Elasticsearch 插件

正如在第 7 章中了解到的，Elasticsearch 的早期版本(5. x 之前的版本)提供了一些插件，这些插件被分为三类：Java 插件、**网站 Site** 插件、**混合 Mixed** 插件。现在的版本中，**网站**插件和**混合**插件官方已不再支持，仅支持 Java 插件。这些 Java 插件必须安装在每个节点上，并且只包含 JAR 文件。在第 7 章中也对 Elasticsearch 插件进行了介绍。

Elastic. co 将插件归为核心插件，这些插件由官方正式开发和维护。社区插件由社区开发和维护。要使用这些插件，需要通过 Elasticsearch 插件程序将其安装到 Elasticsearch 中。核心插件与 Elasticsearch 共同发布，并与 Elasticsearch 版本保持一致。

在本节中，将熟悉一些有意思的插件。可以指定核心插件的名称，将其安装到 Elasticsearch 中：

```
$ ./bin/elasticsearch-plugininstall plugin-name
```

社区插件可以下载安装包，并使用以下命令安装到 Elasticsearch 中：

```
$ ./bin/elasticsearch-plugininstall path/to/plugin/zip/file
```

这个程序可从下列位置安装：本地文件系统、插件 ZIP 包的位置 URL、GitHub 存储库，或 Elasticsearch 核心插件——直接指定插件的名称即可。

要想了解在浏览器中通过什么 URL 来访问插件，可以使用 Cat API 列出所有插件，在

返回结果中会显示每个插件的 URL：

```
GET _cat/plugins
```

有时在 Elasticsearch 运行前，让一个插件已存在或者已被安装，是很有必要的。在这种情况下，可以在 elasticsearch.yml 配置文件中，将一些需要用到的插件设置为必装插件。

例如，如果需要有 EC2 的 Discovery 搜索功能，可将其指定为 Elasticsearch 的必装插件：

```
plugin.mandatory: discovery-ec2
```

可以通过一个以逗号分隔的列表，来指定更多的插件。

要删除插件，只需执行以下命令：

```
$ ./bin/elasticsearch-plugin remove plugin-name
```

这些插件可以进一步按功能来分类，例如 **Alerts**、**Analysis**、**Discovery**、**Ingest**、**Mapper**、**Scripting** 等。本书不能覆盖所有插件类型，但会介绍上述这些类别的某些插件。例如，Discovery 插件可帮助我们使用发现节点的功能并搭建集群。

8.7.1 Discovery 插件

这类插件是非常有用的，因为它们可以使 Elasticsearch 找到不同环境下的集群节点。默认情况下，**Zen Discovery** 被添加到 Elasticsearch 中。此外，还有更多插件可以连接：

- **Azure cloud**：使用 Azure API 以单播方式发现节点，并且也增加了 Azure 作为存储库。
- **GCE cloud**：这是谷歌计算引擎的云服务。
- **AWS cloud**：使用 AWS API 支持 EC2 以单播方式发现节点。

这些插件均由官方正式支持。此外，还有几个社区支持的插件，如 eskka discovery 插件、Kubernetes discovery 插件等。

8.7.2 Ingest 插件

与 Discovery 插件发现节点的设置类似，Ingest 插件可以在 Elasticsearch 中启用 Ingest 节点的功能。其中包含三个插件：

- **Geoip 处理器插件**：使用来自 **Maxmind** 的城市和国家数据库。利用这个数据库，对于文档中的 IP，这个插件会处理其地理位置，然后将详细信息存入 geoip 字段中。
- **用户代理处理器插件**：从请求头信息的 User-Agent 中提取用户代理的相关信息。
- **附件处理器插件**：使用 Tika(Apache 开发的工具包)从多个文件类型中提取元数据

和文本。这个插件使用 Tika,帮助 Elasticsearch 从附件中提取详细信息。常见的附件格式包括 PPT、PDF、XLS 等。

8.7.3 Elasticsearch SQL

有时希望使用关系型数据库视图来查看数据,并且希望使用 SQL 语句在索引中执行操作。该插件为此提供了相应的交互界面,它由社区提供支持。要获取该插件的资源可以访问 https://github.com/NLPchina/elasticsearch-sql。

要安装这个插件,可以执行以下命令:

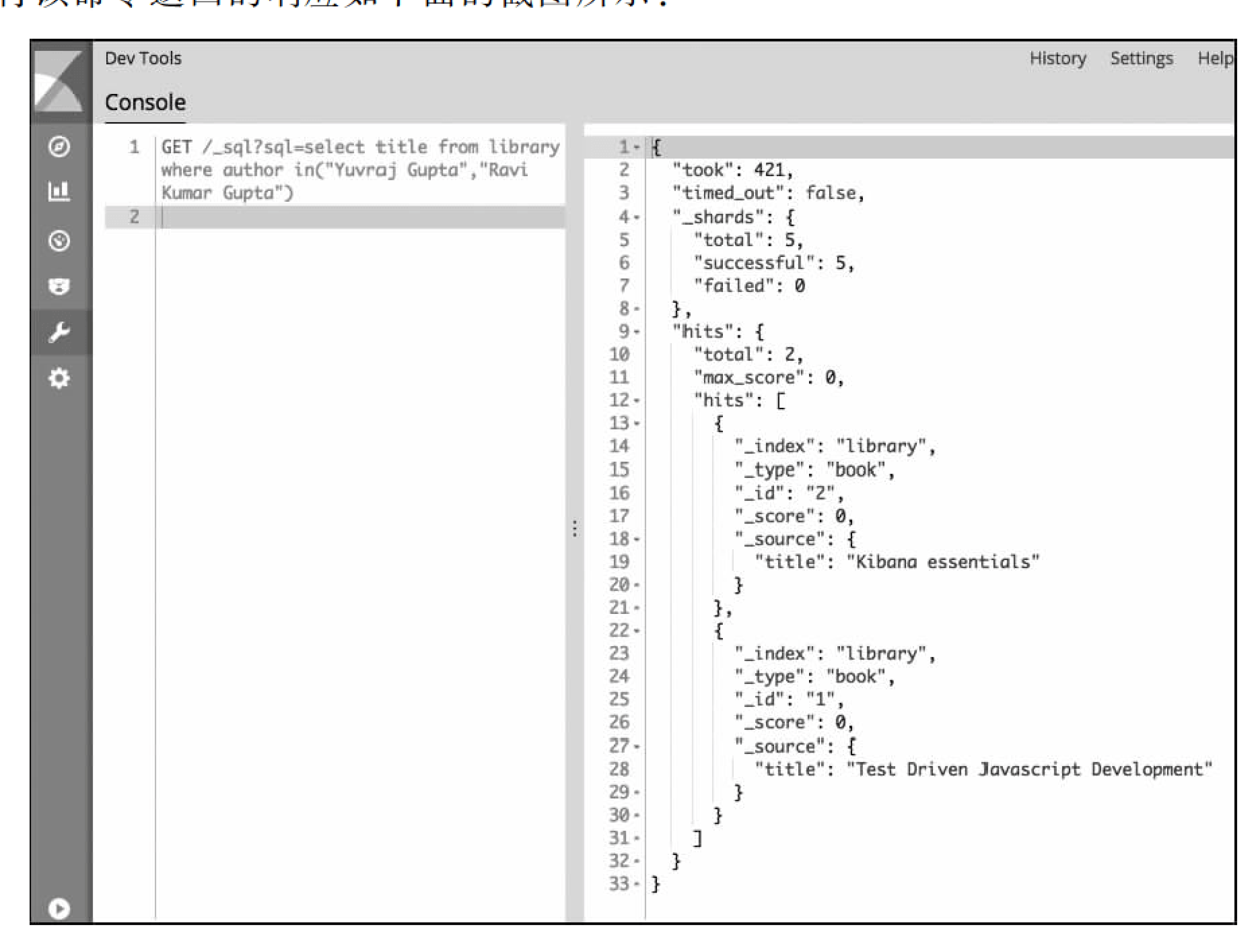
```
$ ./bin/elasticsearch-plugin install
https://github.com/NLPchina/elasticsearch-sql/releases/download/5.1.1.0/
elasticsearch-sql-5.1.1.0.zip
```

网页端插件从 Elasticsearch 中移除时,elasticsearch-sql 不会提供任何单独的交互界面访问和执行 SQL 查询。可以使用控制台(代替网页端插件)运行 SQL 查询。

一个示例的请求如以下命令所示:

```
GET /_sql?sql=select title from library where author in("Yuvraj Gupta","Ravi
Kumar Gupta")
```

执行该命令返回的响应如下面的截图所示:



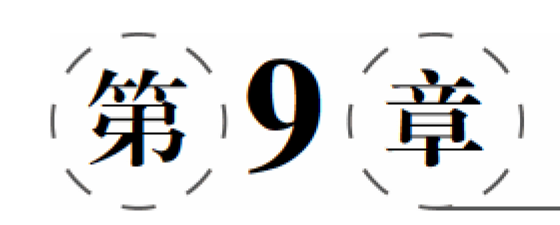
可以发送一个 API 请求，它将显示上述截图中的结果。可以看到，执行结果返回了所有作者名字包含 Yuvraj Gupta 和 Ravi Kumar Gupta 的标题。

> 要了解该插件的更多信息，请访问以下链接：
> https://github.com/NLPchina/elasticsearch-sql/wiki

8.8 本章小结

本章对 Elasticsearch API 进行了介绍，这是一个非常庞大的话题。本章内容不能涵盖所有的 API，但是通过本章内容的学习，我们已经了解到这些 API 如何工作，以及如何借助相应的 API 帮助我们管理 Elasticsearch 集群、节点和索引，甚至如何搜索文档。使用 Kibana 工作时，使用控制台可以做同样的事情。在第 2 章中，我们已经了解了许多基于 REST，使用 HTTP 协议为多种语言和平台开发的客户端。本章还包括其他方面的内容，如调用 Java API 来使用 Transport Client 等。

在下一章中，将重点讨论如何使用插件定制 Elastic Stack。插件使我们能对许多功能进行控制，使我们可以自由地实现一些原本不存在的功能，或者修补现有的功能，并使其能够正常工作。下面将学习创建新插件的方法，并对 Stack 进行个性化定制。



X-Pack 插件中的 Security 与 Monitoring 组件

在前几章中,我们探讨了 Elastic Stack 的四个核心组件——Elasticsearch、Logstash、Kibana 和 Beats。虽然这些组件可传送数据,并完成索引和可视化,但是在生产级别的体系中,仍然有些更为重要的方面值得关注。在谈论生产环境中的服务器时,需要考虑安全性,并希望接收到有关错误、故障等事件的通知。这些重要的特性是 X-Pack 关注的核心。

在本章中,将简要介绍 X-Pack 插件中的各种组件,内容将涵盖组件的功能、安装和配置。

在本章中,将介绍以下主题:

- X-Pack 插件简介;
- X-Pack 插件的安装;
- 探索 Security 组件;
- 查看 X-Pack 插件信息;
- 探索 Monitoring 组件;
- 了解 Profiler。

9.1 X Pack 介绍

学习 X-Pack 插件之前,先了解 X-Pack 插件的是如何出现的。前面已经讨论了 Elastic Stack,在它的早期版本中,Elasticsearch、Logstash、Kibana 和 Beats 组件各自的版本之间存在差异。因此,为了避免用户混淆,Elastic 团队推出了新版 Elastic Stack,其中 Elasticsearch、Logstash、Kibana 和 Beats 组件都是统一的发行版本。使用这些产品之后,终端用户对产品支持会有一些需求,例如 Elasticsearch 集群和 Kibana 授权与认证的需求、使用简单而直观的 UI 来监视 Elasticsearch 集群的需求,以及对强大的报警和通知机制的需求。

X-Pack 是 Elastic Stack 的扩展包。它结合了 Elastic Stack 之外的多种产品，如 Shield、Marvel、Watcher 和 Graph，还提供了额外的 Reporting 功能。X-Pack 与 Elastic Stack 具有相同的发行版本，而不是保留每个产品的不同版本，并且始终检查每个产品的支持矩阵。现在不再需要考虑各种产品及其版本，而只需要安装与 Elasticsearch 对应版本的 X-Pack 插件，即可拥有所有支持、维护和兼容的产品。X-Pack 中的组件可以彼此无缝地配合使用，也可以独立使用，其中提供了设置项、配置和 API 接口，可以通过这些渠道启用或禁用要使用的组件，配置组件以及访问有关组件的信息。

9.2 X Pack 的安装

要安装 X-Pack 插件，需要提前安装 Elasticsearch 5.0 和 Kibana 5.0。此外，必须运行与使用的 Elastic Stack 版本相匹配的 X-Pack 插件。

有关 Elasticsearch 的安装，请参阅第 1 章 **Elastic Stack 概述**中的**安装 Elasticsearch** 部分。有关 Kibana 的安装，请参阅第 1 章 **Elastic Stack 概述**中的**安装 Kibana** 部分。X-Pack 插件将安装在 Elasticsearch 和 Kibana 中，其间将使用 Elasticsearch-plugin 和 Kibana-plugin 程序。

9.2.1 在 Elasticsearch 中安装 X-Pack

要在 Elasticsearch 中安装 X-Pack，请参阅以下步骤：

（1）在 Elasticsearch 安装目录中运行以下命令安装 X-Pack 插件：bin/elasticsearch-plugin install x-pack。

> ℹ 如果使用 DEB / RPM 格式的安装包，请以超级用户身份[①]运行安装。

在安装过程中，系统会要求向 X-Pack 授予额外权限，这是 Watcher 发送电子邮件警报必需的。输入 Y 继续安装，或输入 N 中止安装。

要跳过安装提示符，请使用-b 或--batch 参数安装 X-Pack：

```
bin/elasticsearch-plugin install x-pack --batch
```

（2）X-Pack 会自动为 Security、Monitoring 等组件在 Elasticsearch 中创建索引。因此，如果已禁用自动创建索引，请配置 elasticsearch.yml 以允许 X-Pack 创建索引。在 elasticsearch.yml 中添加以下条目：

---

① 译者注：在 Linux 操作系统中，有些程序需要超级管理员提供执行权限，方可执行。在终端命令开头加上 sudo，执行时输入管理员密码，即可提升权限执行程序。

```
action.auto_create_index:
.security,.monitoring*,.watches,.triggered_watches,.watcher-history*
```

(3) 在 Elasticsearch 安装目录中启动 Elasticsearch。如果已安装 Elasticsearch,则作为服务启动:

```
bin/elasticsearch
sudo service elasticsearch start
```

9.2.2 在 Kibana 中安装 X-Pack

要在 Kibana 中安装 X-Pack 插件,请参阅以下步骤:

(1) 运行以下命令,在 Kibana 安装目录中安装 X-Pack:

```
bin/kibana-plugin install x-pack
```

> 如果使用 DEB/RPM 格式的安装包,请以超级用户身份运行安装。

(2) 从 Kibana 安装目录中启动 Kibana。如果已安装 Kibana,则作为服务启动:

```
bin/kibana
sudo service kibana start
```

要验证 X-Pack 的安装,可在浏览器中输入 Kibana 配置文件中主机字段保存的 URL 或 localhost:5601(默认主机地址),浏览器中将会显示 Kibana 登录界面。

现在你可能想知道如何登录。"我从来没有设置密码,可是现在 Kibana 居然要求我输入密码?"不要担心,默认用户名是"elastic",默认密码是"changeme"①。

> 使用 bin/elasticsearch-plugin 和 bin/kibana-plugin 脚本时,需要 Internet 网络连接才能下载并安装 X-Pack。

9.2.3 在离线系统中安装 X-Pack

要在离线系统中安装 X-Pack 插件,请参阅以下步骤:

(1) 在有 Internet 连接的系统上,从以下 URL 下载 X-Pack 二进制软件包:

```
https://artifacts.elastic.co/downloads/packs/x-pack/x-pack-5.1.1.zip
```

---

① 译者注:Elastic 希望用户在初次使用 X-Pack 插件时尽快修改密码以提高安全性,所以将默认密码设置为 changeme(意为"请把我改了吧")。

(2) 将下载的 ZIP 文件复制或传输到离线系统。

(3) 使用以下命令从 Elasticsearch-plugin 程序安装 X-Pack 插件:

```
bin/elasticsearch-plugin install
file:///home/yuvraj/Downloads/x-pack-5.1.1.zip①
```

> 如果使用 DEB / RPM 格式的安装包,请以超级用户的身份运行安装。

(4) 使用以下命令从 Kibana-plugin 程序安装 X-Pack 插件:

```
bin/kibana-plugin install
file:///home/yuvraj/Downloads/x-pack-5.1.1.zip
```

> 需要在 file://协议之后指定 ZIP 文件的绝对路径。

安装 X-Pack 插件后,所有功能都已安装并启用,如 Shield、Marvel、Watcher、Graph 和 Reporting 等。要在安装 X-Pack 后查看信息,可以使用 X-Pack API,这些将会在后面的“查看 X-Pack 信息”一节中介绍。

> 如果安装 X-Pack 时 Elasticsearch 或 Kibana 已经运行,请重新启动 Elasticsearch 或 Kibana。没有重新启动之前,X-Pack 的设置可能不会立即应用于 Elasticsearch 和 Kibana,这样在使用过程中可能会遇到问题。

9.2.4 卸载 X-Pack

要卸载 X-Pack 插件,请参阅以下步骤:

(1) 关闭 Elasticsearch 和 Kibana。

(2) 在 Elasticsearch 安装目录中运行以下命令,从 Elasticsearch 中删除 X-Pack 插件:

```
bin/elasticsearch-plugin remove x-pack
```

(3) 在 Kibana 安装目录中运行以下命令,从 Kibana 中删除 X-Pack 插件:

```
bin/kibana-plugin remove x-pack
```

> 如果使用 DEB/RPM 格式的安装包,请以超级用户的身份执行此命令。

---

① 译者注:在这行命令中,最后一个参数指定了本地文件系统中一个文件的存储位置,这里“file:///路径”的写法是一种特定的协议书写格式,由“file://”和“/路径”组成。这种路径格式用来提供文件的绝对路径。

（4）启动 Elasticsearch 和 Kibana。

现在探讨 X-Pack 插件中的各个组件。

9.3 Security 组件

X-Pack 插件的 Security 是一种模块。它是针对访问集群数据时所需适当授权机制的需求而创建的，最初是出于 Elasticsearch 集群安全性的需要，目前已经超越了这一点，甚至可以安全地使用 Kibana UI（Kibana 用户界面）并只允许授权用户访问。

安全性通常涉及三 A 问题：

- **认证（Authentication）**：根据用户的身份认证用户。
- **授权（Authorization）**：对于经过身份验证的用户，描述其授权的角色/权限。
- **问责制（Accountability）**：记录用户会话（session）信息、使用信息等。

考虑以上三 A 问题，Elastic 团队使用 Shield 作为 X-Pack 安全模型，以确保三 A 问题被完整涵盖。

对于**认证**，它提供对未授权访问的限制，包括基本密码保护、组织级用户管理和基于 IP 的授权。

对于**授权**，它提供基于角色的访问控制，其中定义了经过身份验证的用户所具有的角色以及基于索引操作的特权。

对于**问责制**，它提供完整的审计跟踪，通过跟踪集群中执行的所有操作和用户级信息，收集与每个用户执行的身份验证和授权相关的信息。

为什么要同时为 Elasticsearch 和 Kibana 安装 X-Pack？为什么要在两者上安装 X-Pack？如果只安装在 Elasticsearch 或 Kibana 上，就不能使用它吗？

这个问题的答案是不行。Elasticsearch 是数据传输/存储层，需要 X-Pack 用于集群/节点相关的配置。作为数据可视化层面的 Kibana 需要 X-Pack 用于不同的功能，例如通过身份验证、报告和管理 X-Pack 的设置来保护 Kibana UI。此外，X-Pack 中的安全模块只有在 Elasticsearch 和 Kibana 上均安装了 X-Pack 时才有效。

在 Elastic Stack 5.0 版本发布之前，使用 Shield 来保护 Elasticsearch 集群，这需要用户使用 REST API 执行操作，例如添加新用户，验证用户，提供权限，对每个用户配置角色，提供对索引的访问，控制访问权限，更改用户密码以及添加用户信息等。通过将 X-Pack 扩展到 Kibana，并在 Kibana 中添加了新的 **Management（管理）**选项卡，现在所有使用 REST API 执行的操作均只需单击下界面中的按钮即可完成配置。

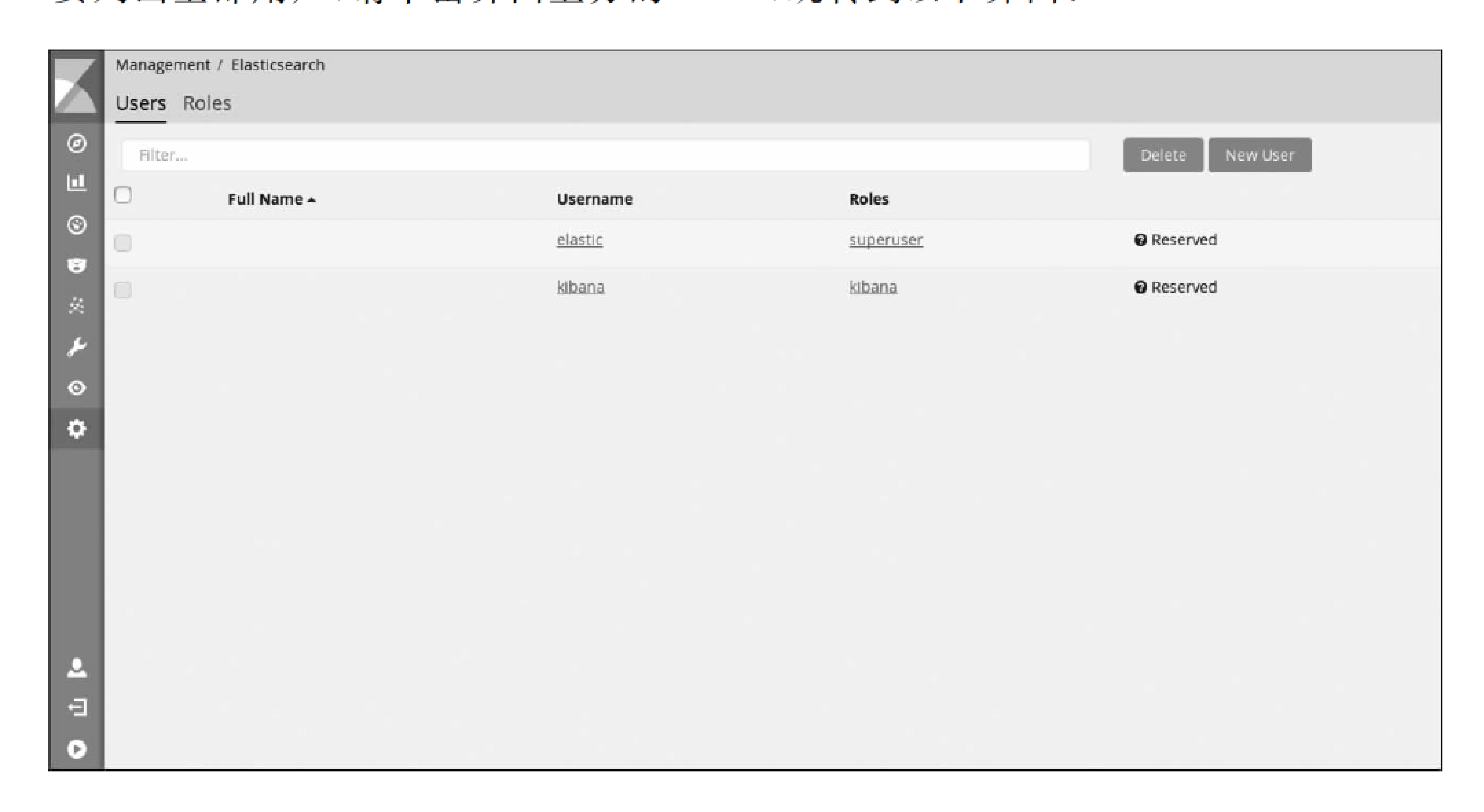
下面探讨如何使用 Kibana UI 和 REST API 创建用户，配置角色以及配置索引级权限。我们将使用 **Management** 选项卡，该选项卡已经在第 4 章 **Kibana 界面**中的**探索设置**

(**Management**)**界面**部分介绍过。

打开 **Management** 选项卡后,可以将其中的选项分为两部分,即 Elasticsearch 和 Kibana。对于与 Shield 相关的安全配置,我们将使用**用户**和**角色**从 Kibana UI 中配置 Shield 的设置项。

9.3.1 列出所有 Security 中的用户

要列出全部用户,请单击界面上方的 **Users**,跳转到以下界面:



在上面的屏幕截图中可以看到,已经创建了两个用户作为默认用户,即 **elastic** 和 **kibana**,它们被配置了不同的角色。

> ℹ 这两个用户都是系统保留的,即不能修改或删除;只能修改它们的密码。

要使用 REST API 列出所有的用户,可以使用以下命令:

```
curl -XGET localhost:9200/_xpack/security/user
```

在上述命令中,localhost 表示主机名,9200 表示端口名称,_xpack 是开放给 API 的 X-Pack 相关设置的参数,security 表示命令中使用了 X-Pack 安全性的 API。

运行上述命令后,会收到报错信息,说明原因如下:

```
"reason" : "missing authentication token for REST request
```

出现问题的原因是对安全性的理解出现错误,也就是说,如果请求来自任何没有指定用户名和密码的用户,将抛出一个错误,因为无法验证用户。

> 即使启用了 Security 功能，也可以允许执行匿名访问用户执行的查询。

因此，要使用 Elasticsearch 上的 REST API 执行任何操作，需要提供用户名，因为已经启用了 X-Pack 中的 Security 功能。

要列出所有使用 REST API 的用户，需要在命令中传递用户名如下：

```
curl -XGET --user elastic localhost:9200/_xpack/security/user
```

系统将要求用户 elastic 输入密码；一经验证，结果中将列出用户。此外，也可以在 REST API 中提供密码以及用户名，方法如下所示：

```
curl -XGET --user elastic:changeme localhost:9200/_xpack/security/user
```

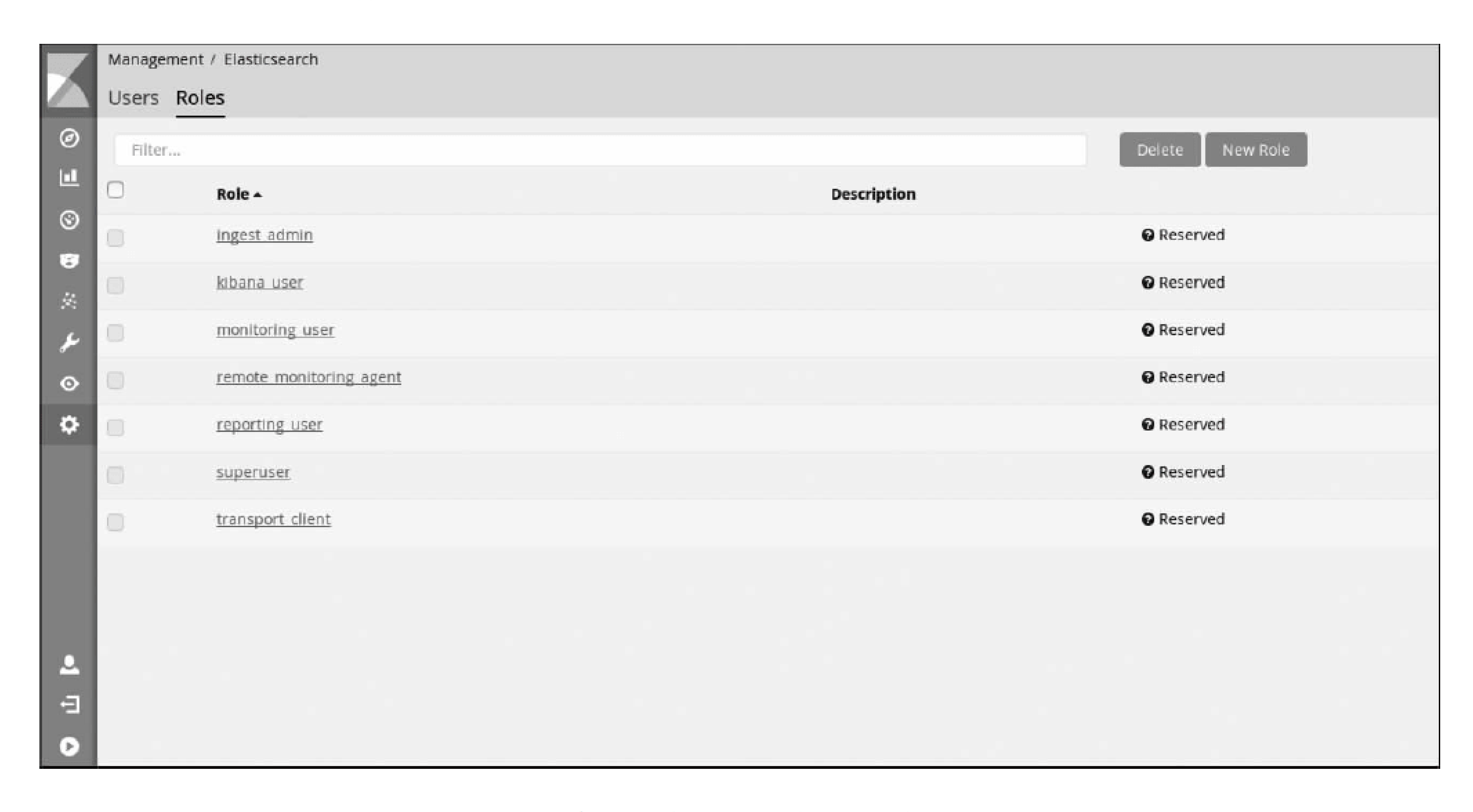
安装 Kibana 之后，Sensitive Console 已经作为 Dev Tools（开发工具）的一部分，可以使用 Sense Console 在 REST API 上运行 curl 命令。该命令的语法如下所示：

```
GET _xpack/security/user
```

为了便于理解，将使用 Sense Console 语法来展示 curl 命令。

9.3.2 列出 Security 中的角色

要列出全部角色，请单击界面上方的 **Roles**，将跳转到以下界面：



在上图中，可以看到已经创建了默认角色，它们是：**ingest_admin**、**kibana_user**、**monitoring_user**、**remote_monitoring_agent**、**reporting_user**、**superuser** 和 **transport_client**。

> 这些默认的角色[①]是系统保留的，即不能修改或删除。

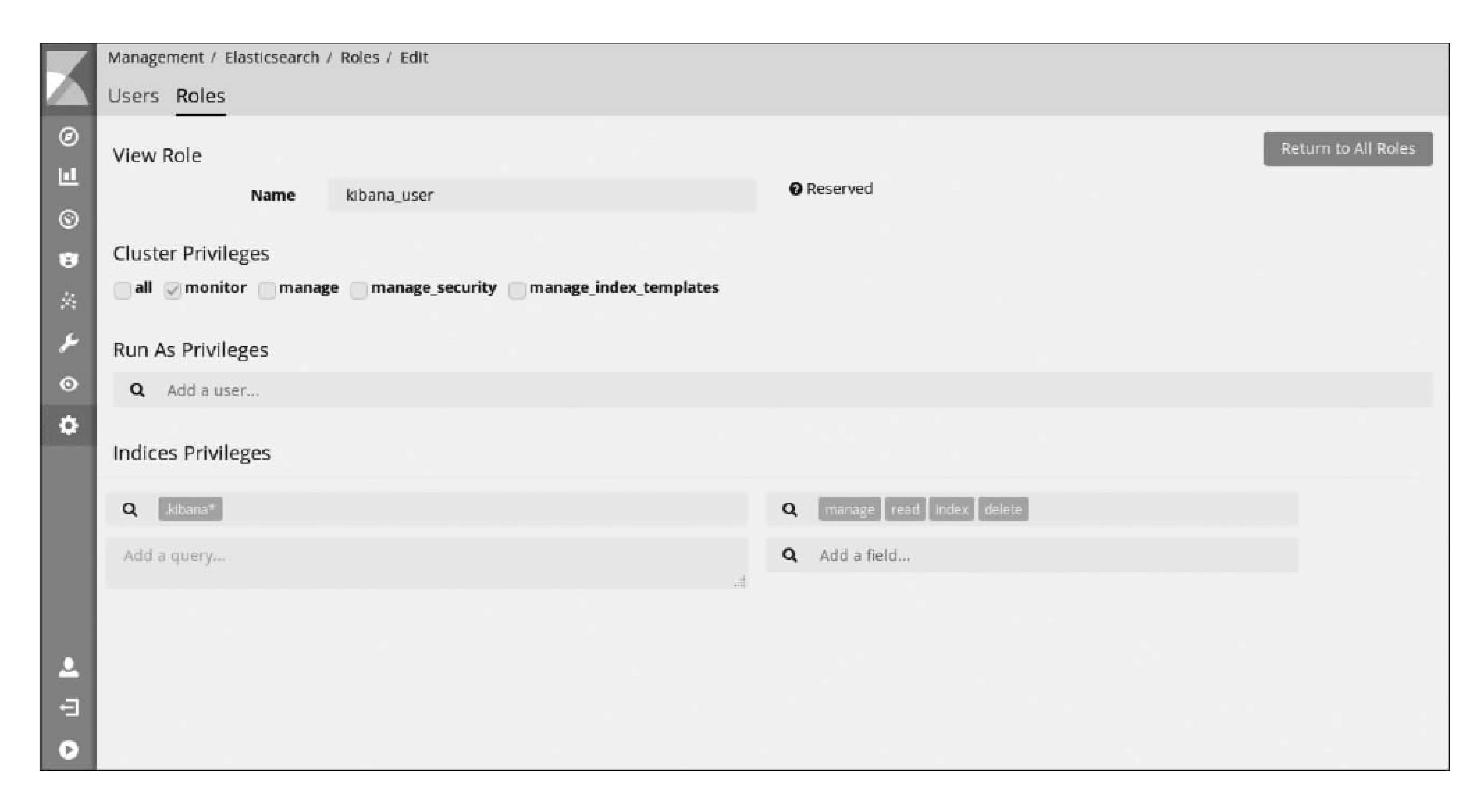
要使用控制台列出角色，请执行如下命令：

```
GET _xpack/security/role
```

角色在这里是十分重要的，因为它们基于分配给每个角色的权限来提供对集群和索引的访问控制。它可以使特定用户按照指定的权限访问集群和索引。

9.3.3 了解 Security 中的角色

单击 **kibana_user** 这一角色，将跳转到以下界面：



在上图中，包含了多种特权条目，例如 **Cluster Privileges**、**Run As Privileges** 和 **Indices Privileges**，看起来十分复杂。让我们来详细了解其中的每个特权。

9.3.3.1 了解 Cluster 集群特权

集群特权提供了在集群上执行各种操作的权限。它们可以分为五种设置，解释如下：

- **all**：提供对集群中可以执行的所有操作的访问权限。通过使用此权限，甚至可以关闭或重新启动节点。
- **monitor**：提供对集群的访问权限，可以从集群中读取数据以进行监视。其中的权限包括但不限于确定集群信息、节点信息、节点(node)/集群(cluster)/快照(snapshot)/恢复

① 译者注：原文中“角色”对应的单词为 rules，笔者认为应为 roles。

(restore)的状态等。

- **manage**：提供 monitor 特权的附加功能，包括执行集群更新操作的权限，例如集群的 rerouting 和 updating 操作。
- **manage_security**：提供对角色和用户的 CRUD(增、删、改、查)操作的访问，以便管理安全性。
- **manage_index_templates**：提供对索引模板(index templates)上可执行的所有操作的访问。

9.3.3.2 了解 Run as 特权

Run as 权限是一种特殊类型的权限，可用于经过身份验证的用户。使用这种权限时，经过身份验证的用户可以代表其他用户提交请求，可以提供或限制访问权限来执行其他用户的操作，而无须重新认证用户。可以指定提供 Run as 特权的用户名。有时，它也被称为模拟(impersonation)，是一种模拟其他用户的方式。

9.3.3.3 了解索引特权

索引(indices)特权为在索引上执行各种操作提供了特权。它们可以被分为以下设置项：

- **All**：这项特权提供对索引执行的所有操作的访问权限。
- **Monitor**：这项特权提供对索引监视所需的所有操作的访问，如索引信息、状态等。
- **Manage**：这项特权提供 monitor 特权的附加功能，它提供了索引管理任务的访问权限，包括但不限于索引设置、使用别名、使用分词器等。
- **Read**：这项特权提供对索引的只读访问权限，例如搜索数据、获取数据，以及其他 actions，如计数、验证、运行 percolator 等。此外，它还提供更新索引映像(mapping)的权限。
- **Index**：这项特权提供对访问索引、更新索引中文档的权限。此外，它还提供了更新映像的权限。
- **Create**：这项特权仅提供对索引文档的访问权限。此外，它还提供更新映像的权限。
- **Delete**：这项特权仅提供删除索引中文档的权限。
- **Write**：这项特权提供所有对索引执行写入操作，例如创建索引、更新和删除文档的权限。它还提供了执行批量操作(如批量插入、处理等)的访问。
- **Delete_index**：这项特权仅提供删除索引的权限。
- **Create_index**：这项特权仅提供创建索引的权限。如果需要在索引中插入文档，那么还需要 write 权限。

• **View_index_metadata**：这项特权提供查看索引元数据的访问权限，如对索引的设置信息（创建日期、分片数、副本数等）和映像（索引字段名称及其数据类型）。

在**索引特权**中，可以用到单个或多个索引的名称、索引的权限、限制用户访问的查询以及限制用户仅查看的字段名称。

我们已经讨论了各种特权的含义，现在尝试解释默认用户的角色，以了解它们提供了哪些权限。

9.3.4　理解默认用户角色

让我们来看看这些默认创建的用户角色，即 **kibana_user**、**superuser** 和 **transport_client**。

9.3.4.1　kibana_user

集群特权被设为 **monitor**，即这一特权仅提供读操作的权限。拥有此权限的角色不允许在集群上进行任何写入、更新、删除操作。

在**索引特权**中，索引的名称被设置为.kibana *，权限被设置为 **manage**、**read**、**index** 和 **delete**。这意味着用户可以使用以.kibana 开头的名称访问索引，并且可以执行诸如监视权限，在索引上的管理操作，对 actions 的读取操作，索引文档，更新索引的映像，对文件的更新和删除等操作。它不能执行诸如对索引写入，创建索引，删除索引以及查看索引的元数据等操作。

9.3.4.2　superuser

集群特权被设置为 **all**，即用户可以对集群执行任何操作。

在**索引特权**中，索引名称被设置为 *，权限被设置为 all。这意味着用户可以访问所有索引，并且可以根据需要在任何索引上执行任何操作。

9.3.4.3　transport_client

这一角色没有被赋予任何集群特权，即用户无法对集群执行任何操作，无论是监视还是管理任务。该角色也没有任何特权访问任何索引。它只拥有在使用 transport API 执行交互时访问所有的字段的权限。

9.3.5　在 Security 中添加新角色

要创建新角色，请转到 Kibana 的 **Management** 选项卡，并单击 **Role**。然后单击 **New Role**（新建角色），单击之后将跳转到指定角色名称和权限的页面。

指定角色的名称，然后选择为集群提供的权限，指定要提供的任何 **Run as 特权**，选择索引的各种权限，如权限后面的索引名称、限制访问权限的查询，以及限制或授予字段访问权

限的字段安全性设置。

> 用户名必须以字母或下画线开头，并且可以包含字母、数字和特殊字符，例如“_”“$”“.”“@”“-”。用户名最少包含 1 个字符，最多可包含 30 个字符。密码必须至少为 6 个字符，最多为 50 个字符。

要使用控制台添加角色，请执行如下命令：

```
POST /_xpack/security/role/testing
{
    "cluster" :["all"],
    "indices" :[
    {
        "names": ["education"],
        "privileges": ["manage","read","index","delete"],
        "query": "{\"match\":{\"State\":\"GA\"}}",
        "fields": ["*"]
    }
    ]
}
```

> 需要将查询中开始和结束双引号以外的双引号进行转义。

执行上述命令将返回以下信息：

```
{
    "role": {
    "created": true
    }
}
```

在上述代码中，testing 是在 POST 中指定的角色名称。

> 可以通过指定 POST 或 PUT 来创建角色，也可以使用 PUT 或 POST 来更新角色。

9.3.6 在 Security 中更新角色

要更新角色的详细信息，例如**集群特权**、**Run as 特权**、**索引特权**、索引名称、查询或字段，请转到 Kibana 中的 **Management** 选项卡，并单击 **Roles**，然后单击要修改的角色名称。

然后，可以更改各种权限，更改后单击 **Save** 保存。

要使用控制台(Console)更新角色,执行如下命令:

```
POST /_xpack/security/role/testing
{
    "cluster" :["all"],
    "indices" :[
    {
    "names": ["education",".kibana"],
    "privileges": ["*"],
    "fields": ["*"]
    }
    ]
}
```

执行上述命令将返回以下信息:

```
{
    "role": {
    "created": false
    }
}
```

9.3.7　了解字段级的 Security

关键字 field_security 提供了授权或限制用户要访问的字段权限的功能。默认情况下,有少量元字段始终为用户提供访问权限,例如_id、_type、_parent、_routing、_timestamp、_ttl、_size 和_index。要使用任何其他元字段,必须显式提供_all,_source 等字段。

可以使用以下表达式来授予或限制访问权限:

- 提供字段名称:直接指定字段的名称,例如"grant":["_all","message"]。
- 使用通配符表达式:指定字段的模式,例如"except" :["date-*"]。
- 访问嵌套字段:读取嵌套字段,例如"grant" :["date.time"],其中 time 是嵌套字段的日期。
- 带嵌套字段的通配符:读取多个嵌套字段,例如"grant" :["date.*"]。
- 将授权与排除(except)合并:指定要访问的字段和用户不能访问的字段,例如:

```
"grant" : [ "date.*" ]
"except": [ "date.time" ]
```

在上面的例子中,将提供访问除 date.time 字段之外的所有以 date 开头的字段。

9.3.8 在 Security 中添加新用户

要创建新用户，请转到 Kibana 的 **Management** 选项卡，并单击 **Users**。然后单击 **New User**，跳转到如下界面：

在上图中可以看到，要创建一个新用户，需要指定用户名、密码、**全名**、电子邮件和**角色**。所有字段的填写都必须使用 Kibana UI。可以根据要求为每个用户指定单个或多个角色。

> ℹ用户名必须以字母或下画线开头，并且可以包含字母、数字和特殊字符，例如"_""$"".""@""-"。用户名最少包含 1 个字符，最多可包含 30 个字符。密码必须至少为 6 个字符，最多为 50 个字符。

使用控制台添加用户，该命令的执行等效于在 Kibana UI 中单击用户全名或用户名的操作，如下所示：

```
POST /_xpack/security/user/test
{
    "password" : "test@123",
    "roles" : [ "kibana_user", "transport_client" ],
    "full_name" : "Testing User",
    "email" : "test@test.com"
}
```

> ℹ可以通过指定 POST 或 PUT 来创建用户。要更新除用户名和密码之外的用户信息，请使用 PUT 或 POST 更新详细信息。要更新密码可以使用 Reset Password API。

执行上述命令将返回以下内容：

```
{
    "user": {
    "created": true
    }
}
```

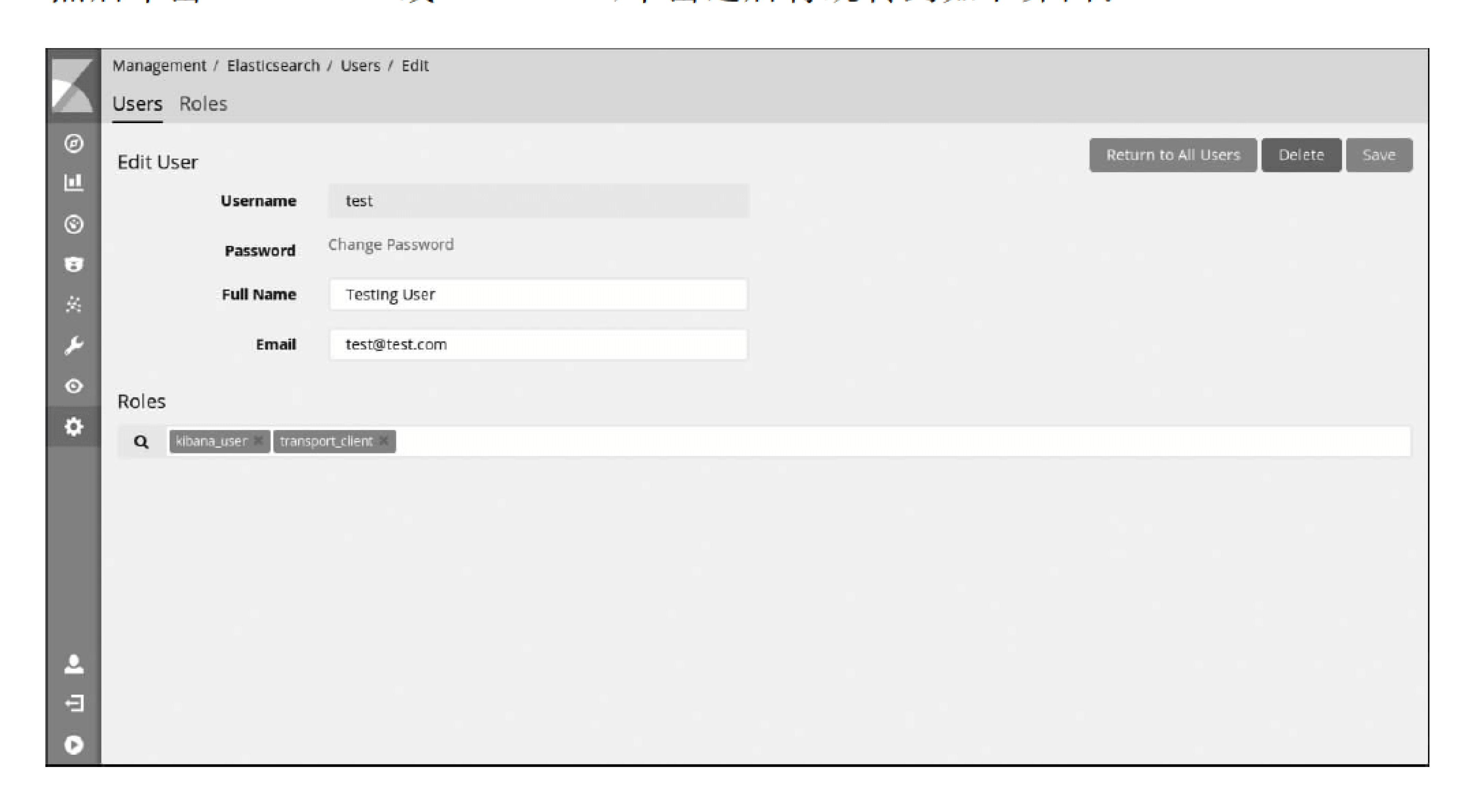
在上述代码中，test 是与 POST 一起指定的用户名。

> 要对用户执行 CRUD（增、删、改、查）操作，必须至少具有一个 manage_security 集群特权。

9.3.9 在 Security 中更新用户详细信息

要更新用户的详细信息，如密码、全名、电子邮件和角色，请转到 Kibana 的 **Management** 选项卡，并单击 **Users**。

然后单击 **Full Name** 或 **Username**，单击之后将跳转到如下界面：



如上图所示，可以更改用户的密码、**全名**、**电子邮件**和**角色**。进行更改后，可以单击**保存**。

要使用控制台更新用户的详细信息，请执行如下所示的命令：

```
PUT /_xpack/security/user/test
{
    "password" : "test@123",
    "roles" : [ "kibana_user", "transport_client" ],
```

```
    "full_name" : "Test User",
    "email" : "test@test.co.in"
}
```

执行该命令将返回如下内容：

```
{
    "user": {
    "created": false
    }
}
```

9.3.10 在 Security 中修改用户密码

可以使用 **Management** 选项卡更改用户的密码，如前面**在 Security 中更新用户详细信息**部分所述。

要使用控制台更改用户的密码，执行如下命令：

```
PUT /_xpack/security/user/test/_password
{
    "password" : "test123"
}
```

执行此命令将返回如下内容：

```
{ }
```

会收到一个空的回复，表示密码已被更改。

9.3.11 在 Security 中删除角色

要删除角色，请转到 Kibana 的 **Management** 选项卡，并单击 **Roles**。接着选中要删除的角色名称旁边的复选框，然后单击 **Delete** 按钮，将要求确认删除所选的角色。

要使用控制台删除角色，执行以下命令：

```
DELETE /_xpack/security/role/testing
```

9.3.12 在 Security 中删除用户

要删除用户，请转到 Kibana 的 **Management** 选项卡，并单击 **Users**。接着选中要删除的用户名旁边的复选框，然后单击 **Delete** 按钮，将要求确认删除所选的用户。

要使用控制台删除角色，执行以下命令：

```
DELETE /_xpack/security/user/test
```

了解 Security 组件之后，让我们来查看 X-Pack 插件的信息。

9.4　查看 X Pack 信息

X-Pack 插件提供了一个 API，可以使用它来查看插件的信息。它提供了构建细节、许可证详细信息以及 X-Pack 插件中每个组件的详细信息。

要在 Kibana UI 中使用控制台查看信息，执行如下命令：

```
GET /_xpack
```

执行此命令将返回如下响应信息：

```
{
    "build": {
        "hash": "821d294",
        "date": "2016-12-06T13:09:18.057Z"
    },
    "license": {
        "uid": "e065e495-3fb4-4cb5-8233-263074ee57e7",
        "type": "trial",
        "mode": "trial",
        "status": "active",
        "expiry_date_in_millis": 1485882488285
    },
    "features": {
    "graph": {
        "description": "Graph Data Exploration for the Elastic Stack",
        "available": true,
        "enabled": true
    },
    "monitoring": {
        "description": "Monitoring for the Elastic Stack",
        "available": true,
        "enabled": true
    },
    "security": {
        "description": "Security for the Elastic Stack",
        "available": true,
        "enabled": true
    },
```

```
    "watcher": {
        "description": "Alerting, Notification and Automation for the Elastic
        Stack",
        "available": true,
        "enabled": true
    }
    },
    "tagline": "You know, for X"
}
```

结果中提供了有关组件的信息以及启用的字段，这意味着可以灵活地启用或禁用想要使用的功能。

要获取 X-Pack 插件许可证的详细信息，执行如下命令：

```
GET /_xpack/license
```

执行此命令将返回如下响应信息：

```
{
    "license": {
    "status": "active",
    "uid": "e065e495-3fb4-4cb5-8233-263074ee57e7",
    "type": "trial",
    "issue_date": " 2017-01-01T17:08:08.285Z",
    "issue_date_in_millis": 1483290488285,
    "expiry_date": " 2017-01-31T17:08:08.285Z",
    "expiry_date_in_millis": 1485882488285,
    "max_nodes": 1000,
    "issued_to": "elasticsearch",
    "issuer": "elasticsearch",
    "start_date_in_millis": -1
    }
}
```

下面来看看如何通过配置来启用或禁用其中的功能。

要启用或禁用 X-Pack 插件的特征功能，请将以下设置添加到 elasticsearch.yml 和 kibana.yml 配置文件中：

```
xpack.security.enabled
xpack.monitoring.enabled
xpack.graph.enabled
xpack.watcher.enabled
xpack.reporting.enabled
```

将上述设置值设置为 false 将禁用相应的功能，同样，将其设置为 true 将启用它们。下面我们来探讨 X-Pack 插件中的其他组件。

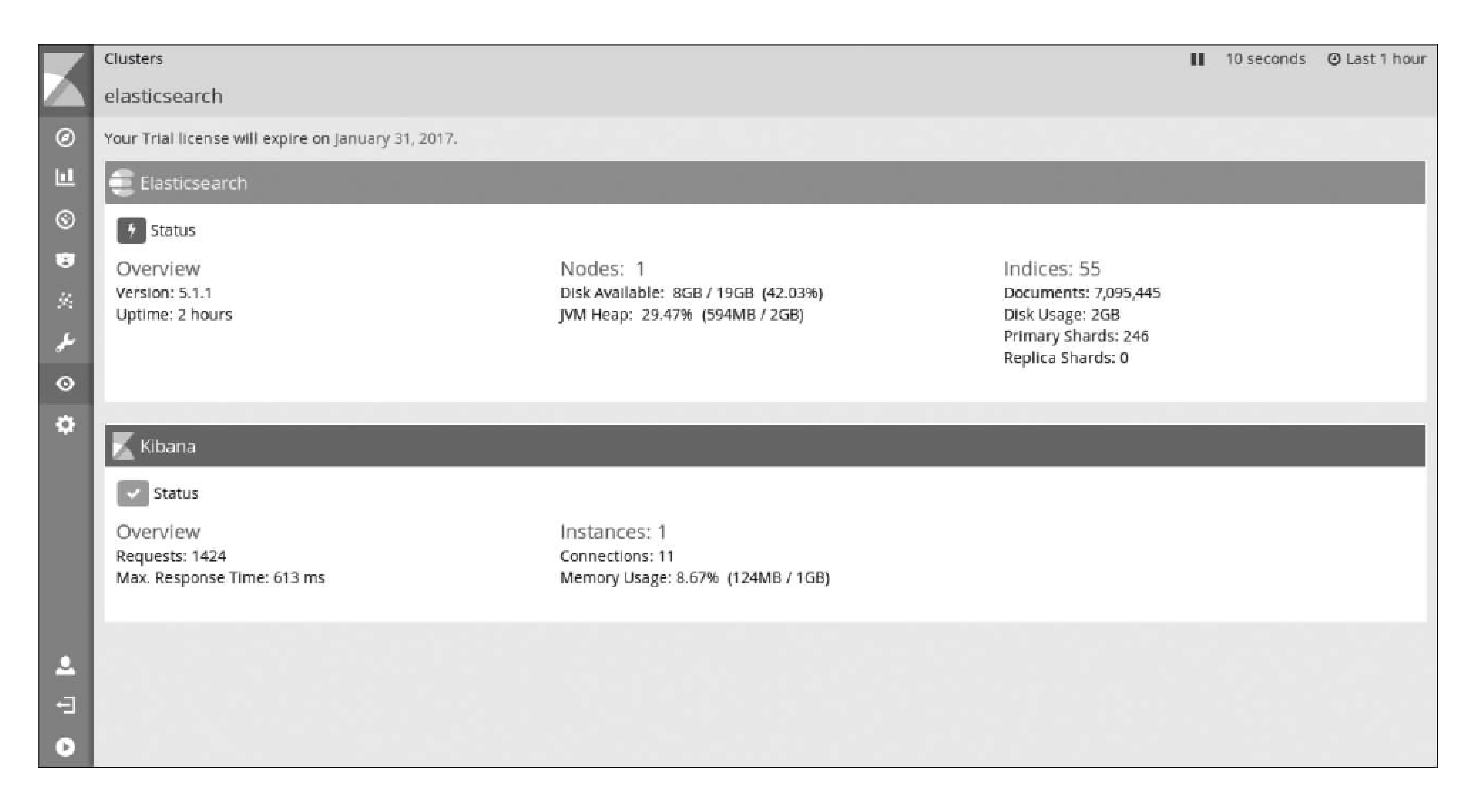
9.5 Monitoring 组件

Monitoring 是 X-Pack 插件中的另一个组件，它已经不再需要使用 UI 来监视 Elasticsearch 的集群、索引和节点。它提供了每个集群、集群中每个索引以及每个节点的详细统计信息。

将 Monitoring 组件纳入 X-Pack，就可以直接从 Kibana 访问 Monitoring UI，而无须从 Kibana 界面跳转至其他界面。此前，它被称为 Marvel，可提供 UI 来监视和获取 Elasticsearch 集群、节点和索引的统计信息，甚至可以看到正在运行的 Kibana 实例(instance)的性能，这是一个附加功能。

Monitoring 由两个子组件构成：监控代理(monitoring agent)和监控程序(monitoring application)。监控代理将安装在要获取统计信息的每个节点上，并使用 Kibana 中的 Monitoring 面板收集 Elasticsearch 中用来执行可视化的数据和索引。它存在于 Elasticsearch 的 X-Pack 插件中，分别安装在每个节点上。此外，监控代理可以灵活地对同一个集群或不同集群上的数据创建索引。监控程序提供用户界面来查看统计信息。它同时也存在于 Kibana 的 X-Pack 插件中，被安装在所要查看信息的机器上。

下面介绍 X-Pack 插件中 Monitoring 组件提供的各种统计信息。要查看 Monitoring 组件，可使用 Kibana 左侧导航窗格中的 **Monitoring** 选项。跳转后，将看到以下界面：



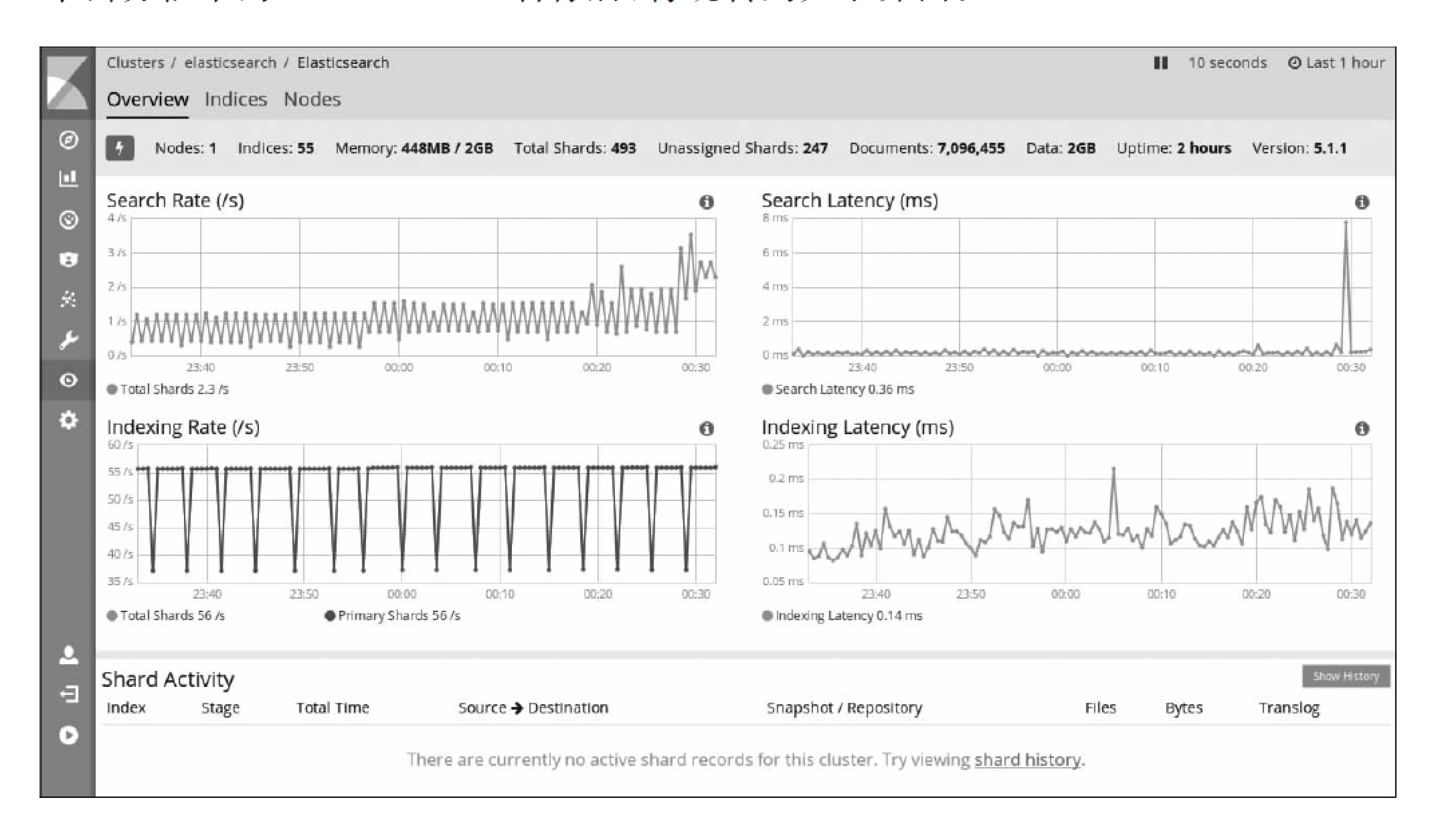
在上图中可以注意到一些信息。在顶部窗格中可以看到，默认情况下，监视 UI 中的信

息每10秒更新一次，正在查看最近每1小时的数据。此外，可以看到集群名称，在这里只有一个，即**Elasticsearch**。在页面的下半部分，可以看到两个方框，一个用于显示**Elasticsearch**的信息，而另一个显示**Kibana**信息，它提供Monitoring组件返回的统计信息。Elasticsearch的方框中给出了相关的简要信息，例如集群状态(Status)、集群正常运行时间(Uptime)、可用磁盘空间总容量(Disk Available)、JVM堆占用百分比(JVM Heap)和索引数量(Indices)，以及其他信息如文件数(Documents)、用于存储文档的分片总数(Primary Shards/Replica Shards)和磁盘空间总容量(Disk Usage)。此外，还可以看到使用的X-Pack许可证的有效期。对于X-Pack的试用版，可以免费使用30天。Kibana的方框给出了Kibana状态的简要信息，包括Kibana正在运行的实例数量(Instances)以及其他信息，如请求总数(Requests)、连接数(Connections)、内存占用百分比(Memory Usage)和最大响应时间(Max. Response Time)。

我们来看看各种可用的Elasticsearch和Kibana监控信息。

9.5.1 探索 Elasticsearch 的监控统计

单击方框中的Elasticsearch名称后，将跳转到如下界面：



在上图中可以看到，Monitoring的用户界面看起来十分简洁优雅。界面中一次性提供了所有相关信息。它包含三个选项卡，即**总览(Overview)**、**索引(Indices)**和**节点(Nodes)**。

9.5.1.1 了解总览选项卡

总览选项卡是打开Elasticsearch监控模块时的默认选项卡。在选项卡的名称下面，给出

了重要的信息，例如状态、节点数量(Nodes)和索引数量(Indices)、占用的内存容量(Memory)、分片总数(Total Shards)、未分配分片总数(Unassigned Shards)、索引中的文档总数(Documents)、用于存储文档的磁盘空间使用量(Data)、正常运行时间(Uptime)和 Elasticsearch 版本(Version)。提供相关信息的顶部窗格在**总览**、**索引**和**节点**选项卡中是通用的。

在上图中各项性能指标的下面，有一些很好看的图表，它们展示了每秒的搜索速率(Search Rate)、每 ms(毫秒)的搜索延迟(Search Latency)、每秒的索引率(Indexing Rate)、每 ms 的索引延迟(Indexing Latency)以及分片活动(Shard Activity)。它默认给出了过去一小时的信息。我们可以更改查看数据和自动刷新页面的设置。图表下方右侧的**显示历史记录(Show History)**按钮可用来查看分片处理记录。

9.5.1.2　了解索引选项卡

单击**索引**选项卡后，会看到以下界面：

Clusters / elasticsearch / Elasticsearch　　10 seconds　Last 1 hour

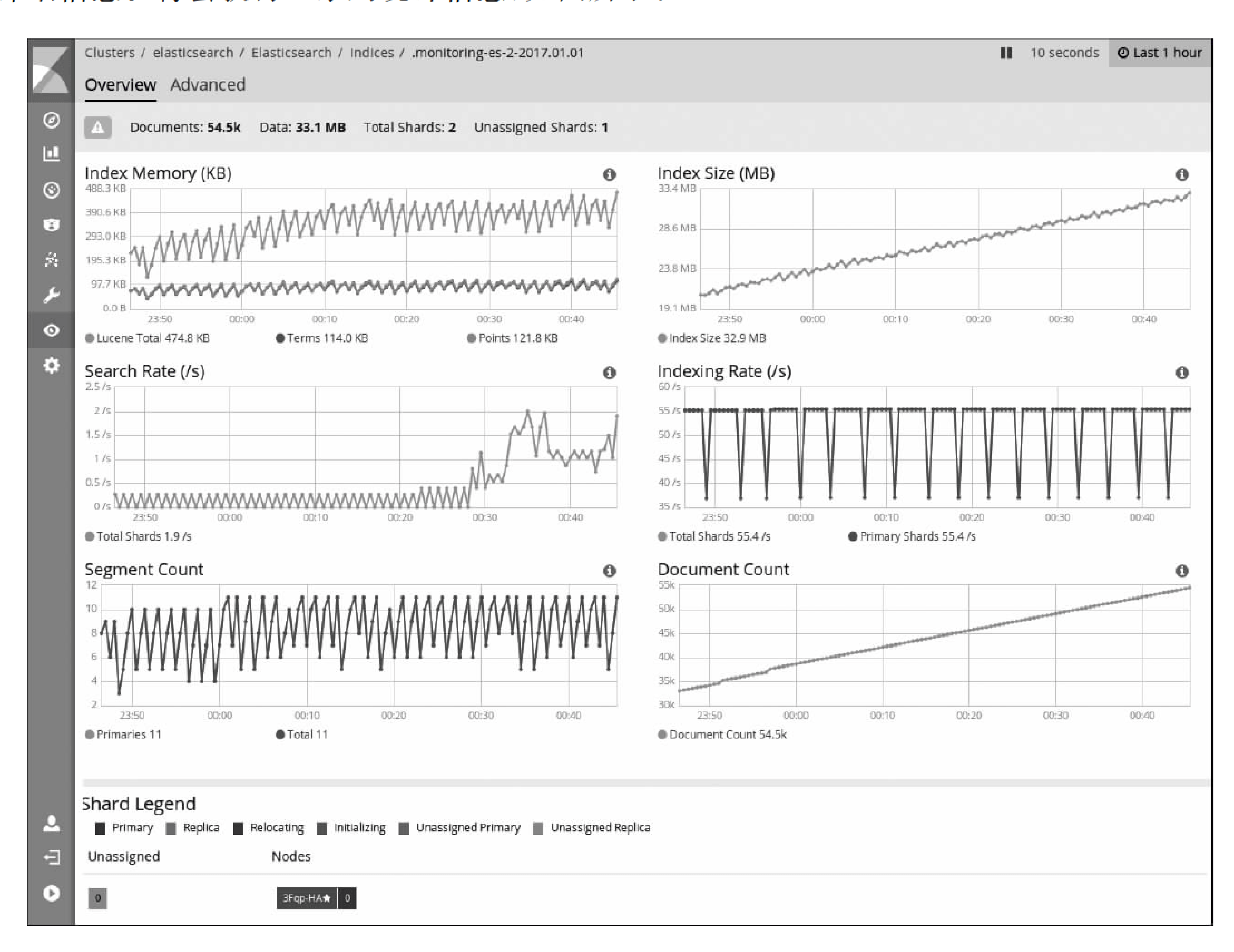
Overview　Indices　Nodes

Nodes: 1　Indices: 55　Memory: 647MB / 2GB　Total Shards: 493　Unassigned Shards: 247　Documents: 7,098,599　Data: 2GB　Uptime: 2 hours　Version: 5.1.1

Indices　Filter Indices　20 of 48　Show system indices

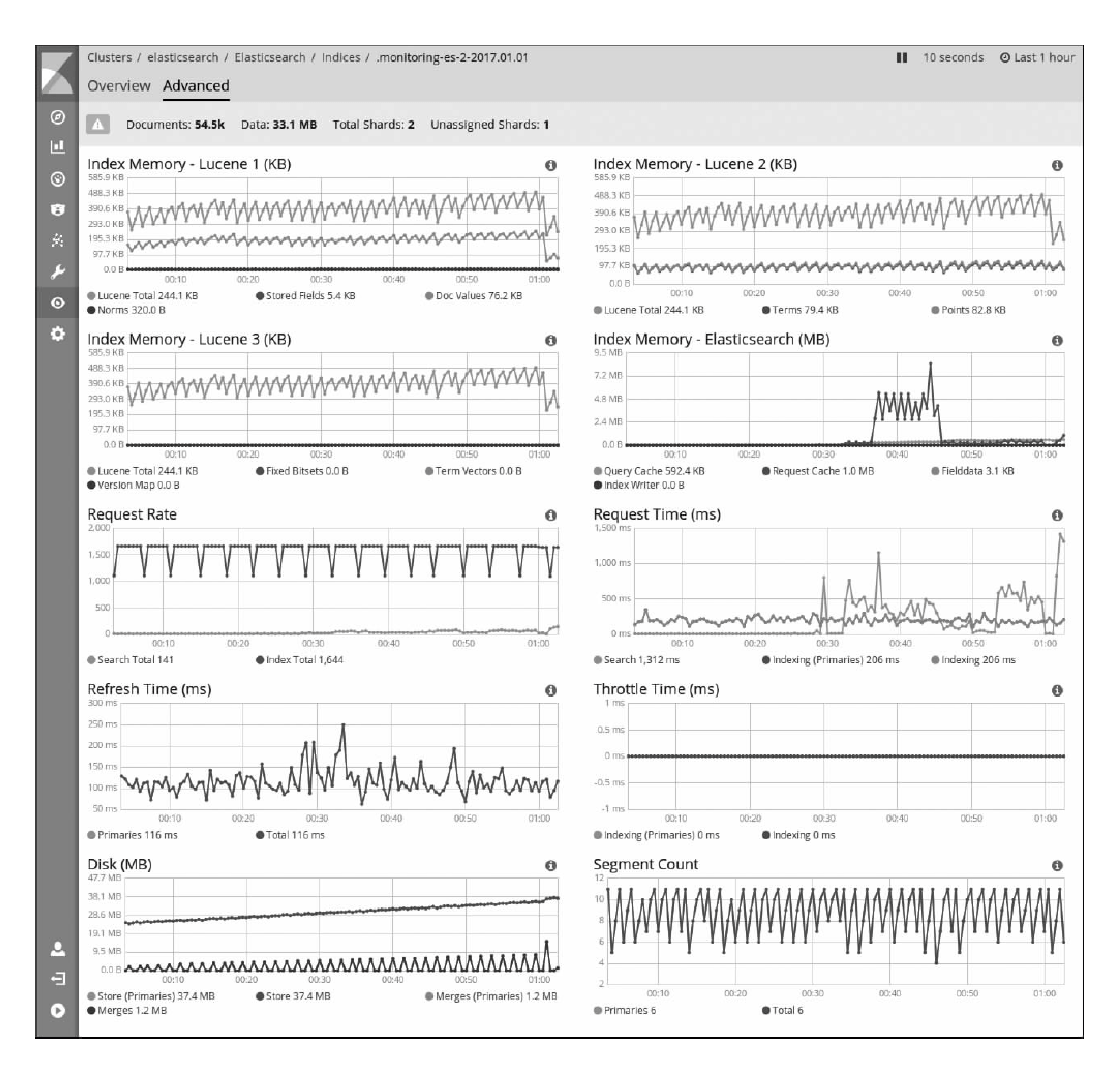
Name	Status	Document Count	Data	Index Rate	Search Rate	Unassigned Shards
packetbeat-2016.12.22		40.8k	10.7 MB	0 /s	0 /s	6
topbeat-2017.01.01		100k	28.2 MB	0 /s	0 /s	5
topbeat-2016.12.26		295	295.2 KB	0 /s	0 /s	5
topbeat-2016.12.22		148.3k	42.3 MB	0 /s	0 /s	5
topbeat-2016.12.21		568.8k	160.4 MB	0 /s	0 /s	5
topbeat-2016.12.20		108.6k	30.6 MB	0 /s	0 /s	5
topbeat-2016.12.19		10.1k	3.4 MB	0 /s	0 /s	5
topbeat-2016.12.18		496.9k	135.8 MB	0 /s	0 /s	5
school		274	190.3 KB	0 /s	0 /s	5
sales		2.4k	1.3 MB	0 /s	0 /s	5
packetbeat-2017.01.01		20.2k	6.7 MB	0 /s	0 /s	5
packetbeat-2016.12.29		113.6k	35.8 MB	0 /s	0 /s	5
packetbeat-2016.12.28		25.2k	8.5 MB	0 /s	0 /s	5
packetbeat-2016.12.26		577	647.6 KB	0 /s	0 /s	5

如上图所示，我们获取了存储在集群中的各种索引信息。它展示了各种信息，如索引名称 **Name**、每个索引中的文档总数（Documents）、存储文件占用的磁盘空间（Data）、每秒的索引速率（Indexing Rate）、每秒的搜索速率（Search Rate）和每个索引中未分配的分片数（Unassigned Shards），还可以搜索索引或按名称过滤索引，以及选中“显示系统索引”（Show system indices）复选框来查看系统创建的索引；甚至可以单击索引名称来查看每个索引的详细信息。将会获得一系列统计信息，如图所示：



在上图中，可以查看顶部窗格中的信息，例如索引的状态、索引中的文档总数（Documents）、用于存储文档的磁盘空间（Data）、分片总数（Total Shards）和未分配的分片数量（Unassigned Shards）。下面我们查看以 KB 为单位的索引所占内存容量（Index Memory）、每秒的搜索速率（Search Rate）、每秒的索引率（Indexing Rate）、索引大小（Index Size）、文档总数（Document Count）和分段总数（Segment Count）等可视化内容。此外，还可以看到节点的名称和节点间的分片状态。

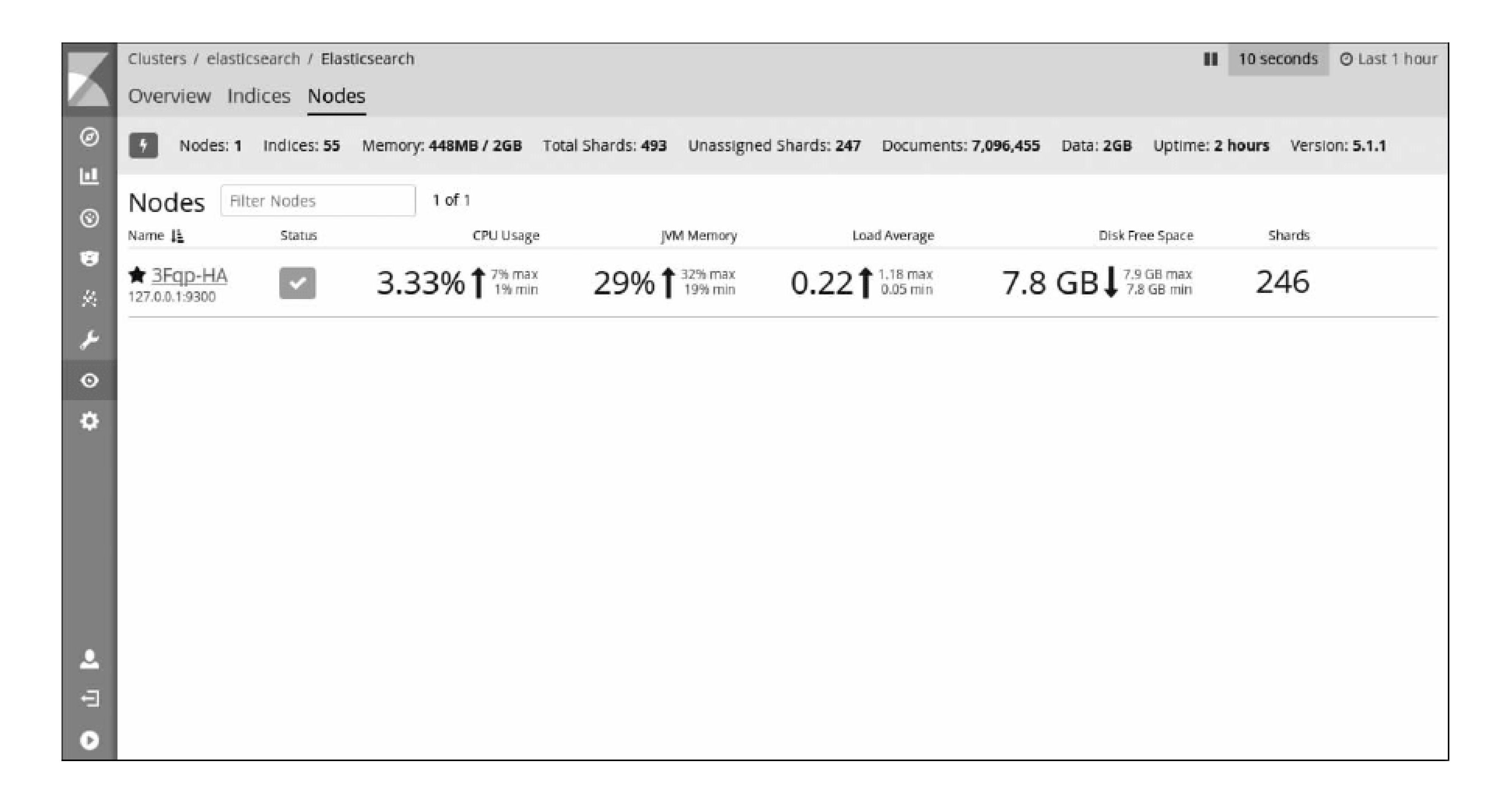
单击 **Advanced** 高级选项卡来获取特定索引的详细信息，将看到如下图所示的统计信息：



在上图中，可以查看顶部窗格中的信息，例如索引的状态、索引中的文档总数(Documents)、用于存储文档的磁盘空间(Data)、分片总数(Total Shards)和未分配的分片数量(Unassigned Shards)。在下面我们可以可视化 Lucene 的索引内存占用量(Index Memory of Lucene)、每 KB 的 Elasticsearch 索引内存占用量(Index Memory)、请求速率(Request Rate)、每毫秒的请求时间(Request Time)、每毫秒的刷新时间(Refresh Time)、每毫秒的节流时间(Throttle Time)和每 MB 的磁盘使用量(Disk)和分段总数(Segment Count)。

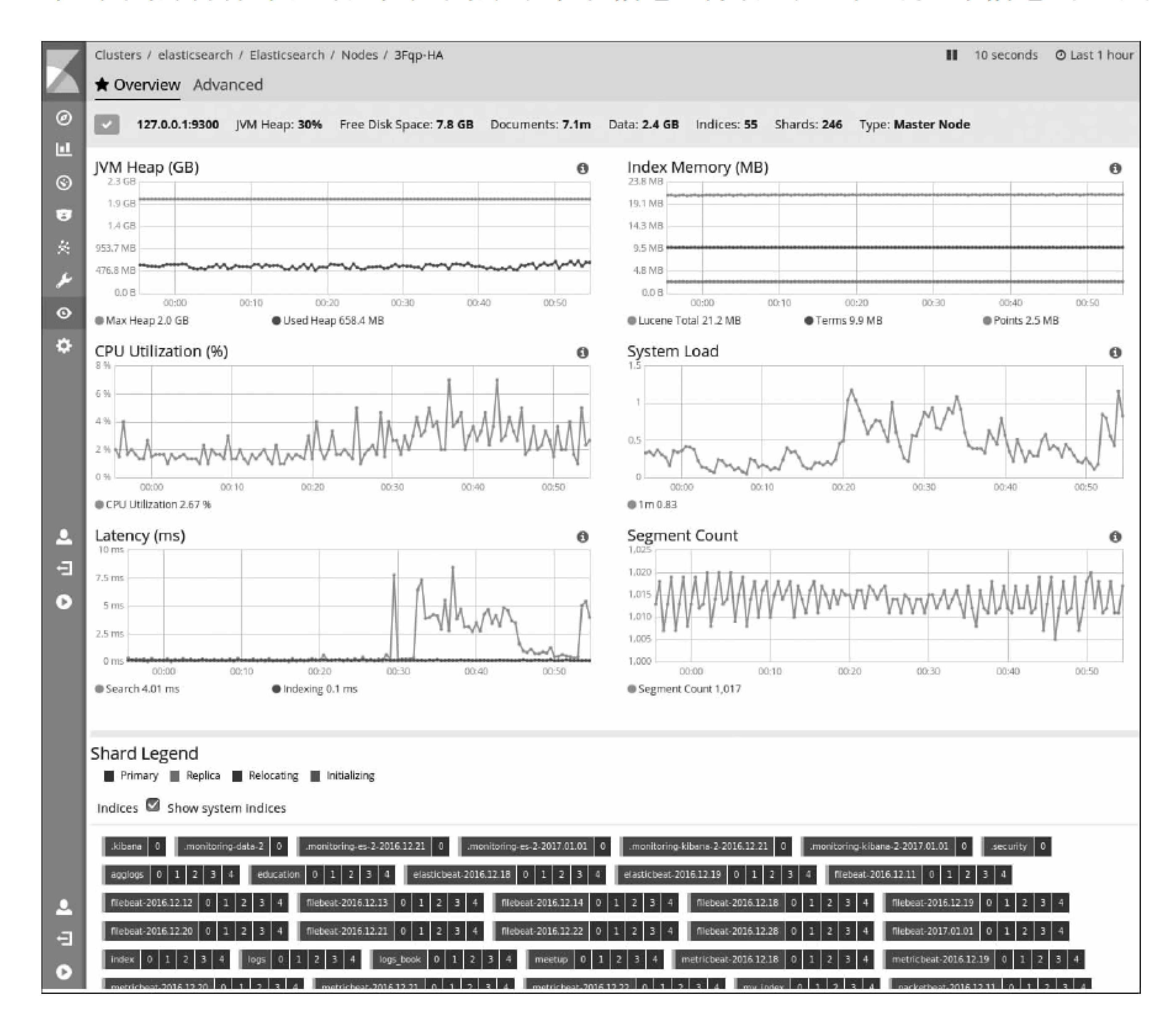
9.5.1.3　了解节点选项卡

单击**节点**选项卡后，会看到以下界面：



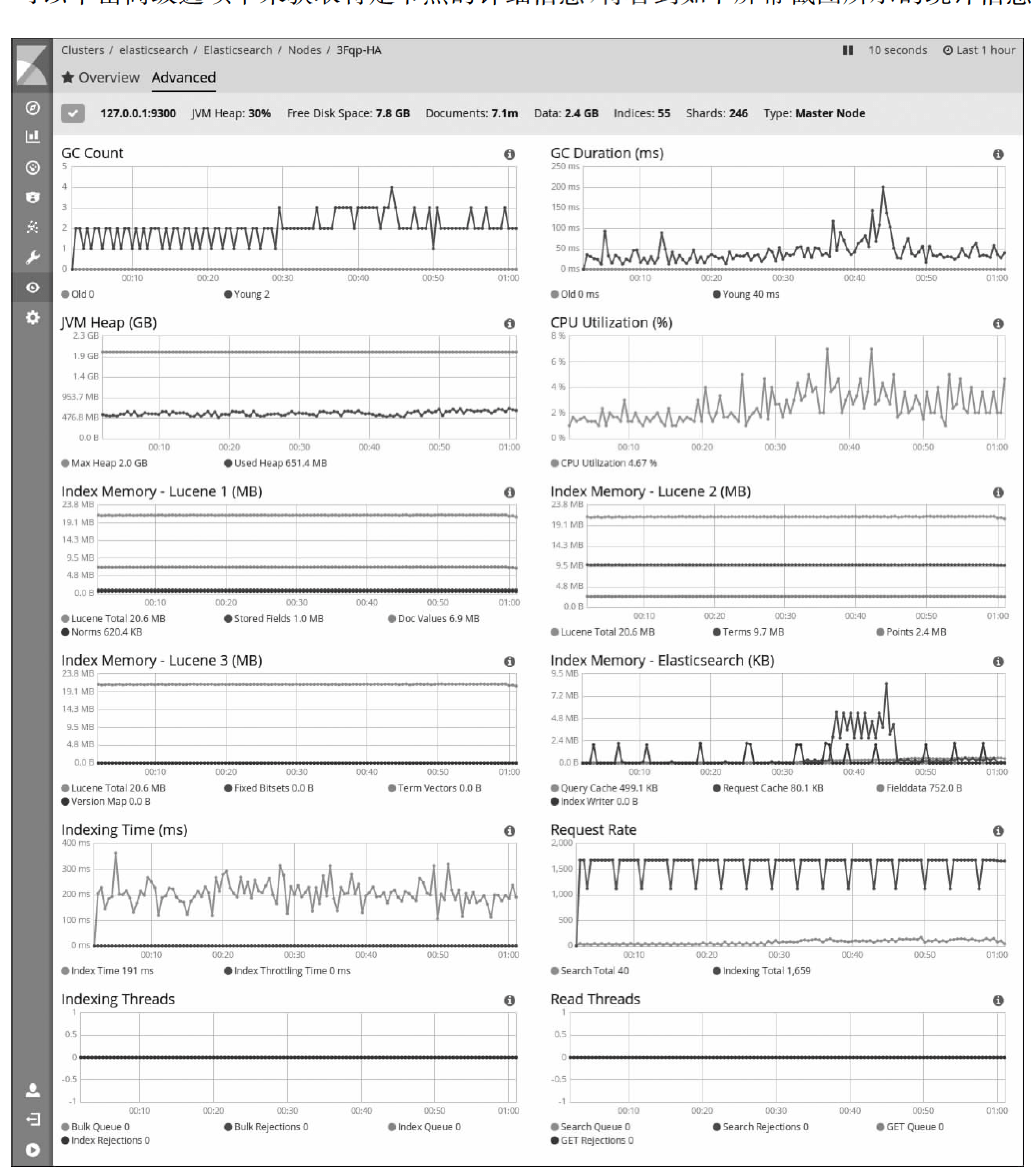
如上图所示，可以得到集群中各种节点的信息。它为我们提供了诸如节点名称、节点状态、CPU 使用情况(**CPU Usage**，每个节点的平均、最小和最大使用量)、JVM 内存(**JVM Memory**，平均、最小和最大使用量)的信息、平均负载(Load Average，平均、最小和最大使用量)、磁盘可用空间(**Disk Free Space**，平均、最小和最大使用量)以及每个节点中已分配的分片总数(Shards)。

可以单击节点名称来查看每个节点的详细信息，将看到一系列统计信息，如下图所示。



在该图中，可以查看顶部窗格中的信息，例如节点的状态、节点的主机 IP 地址、JVM 堆占用百分比(JVM Heap)、可用磁盘空间(Free Disk Space)、文档总数(Documents)、用于存储文档的磁盘空间(Data)、节点中的索引总数(Indices)、分片的总数(Shards)和节点的类型(Type)。下面我们可以可视化查看每 GB 的 JVM 堆使用量(JVM Heap)、每 MB 的索引内存占用量(Index Memory)、CPU 利用率百分比(CPU Utilization)、系统平均负载(System Load)、每秒延迟(Latency)和索引分段数量(Segment Count)等内容。此外，还可以看到各种索引的名称和分片状态。

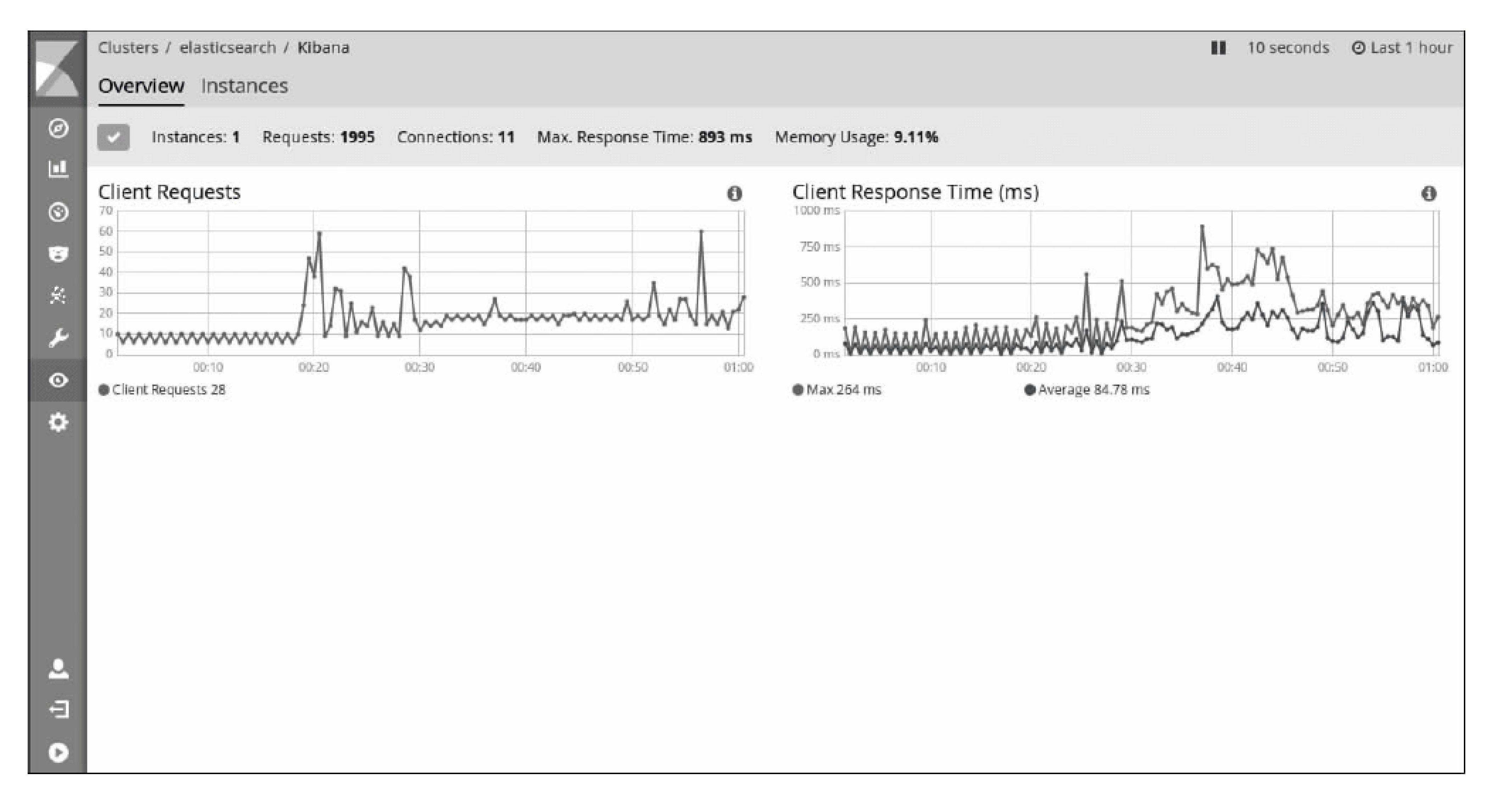
可以单击高级选项卡来获取特定节点的详细信息，将看到如下屏幕截图所示的统计信息：



在上图中，可以查看顶部窗格中的信息，例如节点的状态、节点的主机 IP 地址、JVM 堆占用百分比(JVM Heap)、可用磁盘空间(Free Disk Space)、文档总数(Documents)、用于存储文档的磁盘空间(Data)、节点中的索引总数(Indices)、分片的总数(Shards)和节点的类型(Type)。下面可以查看 GC 计数(GC Count)[①]、每毫秒的 GC 持续时间(GC Duration)、每 GB 的 JVM 堆占用大小(JVM Heap)、CPU 利用率百分比(CPU Utilization)、Lucene 使用的索引内存容量(Index Memory used byLucene)、Elasticsearch 使用的索引内存容量(Index Memory used by Elasticsearch)、每毫秒的索引时间(Indexing Time)、请求速率(Request Rate)、索引和读取线程数(Indexing/Read Threads)。

9.5.2 探索 Kibana 的监控统计

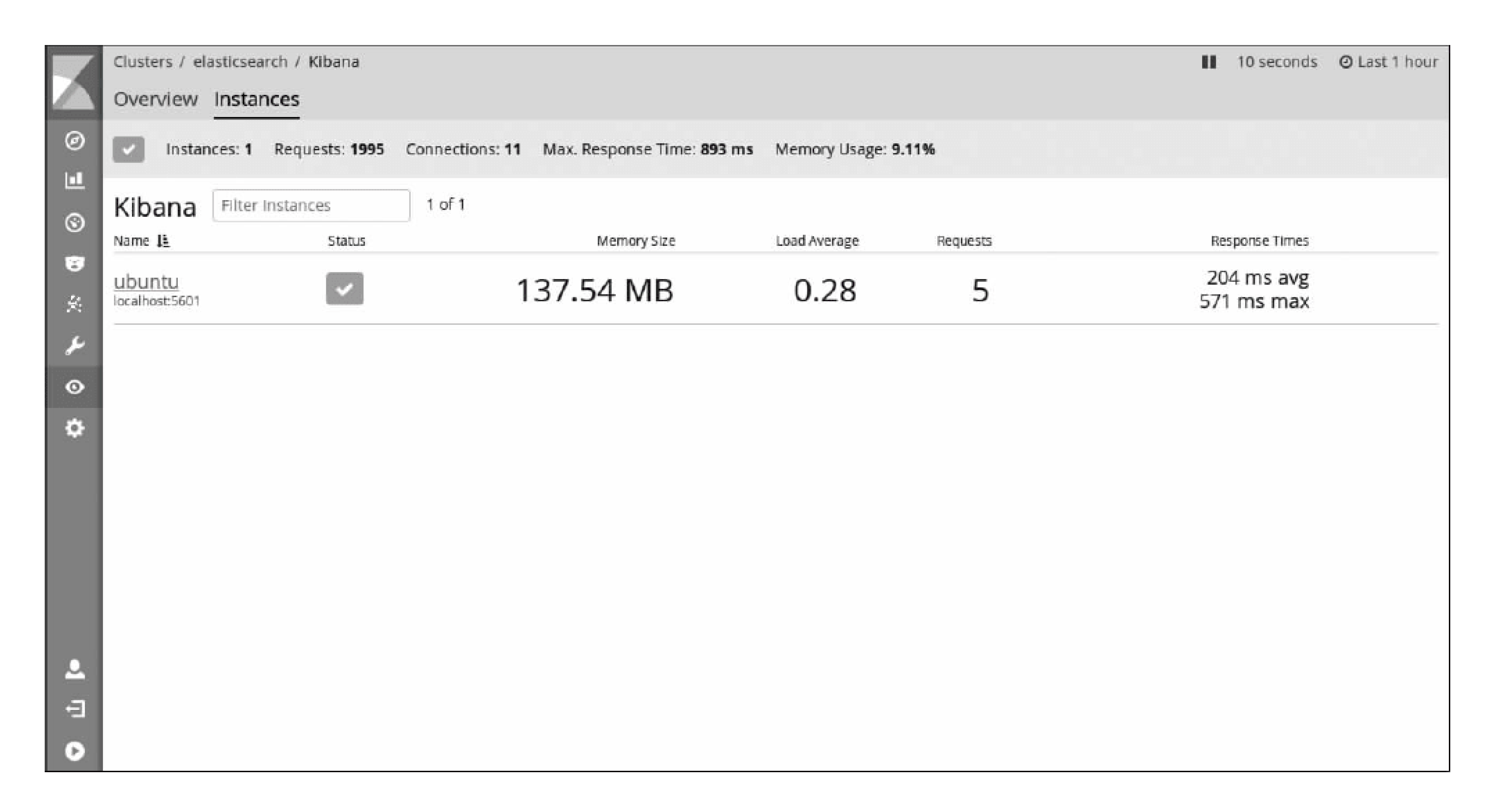
在方框中单击 Kibana 名称，将会看到以下界面：



在上图中可以看到，Monitoring 的界面看起来十分简洁优雅，界面中一次性提供了所有相关信息。在顶部窗格中，给出了实例的状态、运行的实例总数(Instances)、请求总数(Requests)、总连接数(Connections)、最大响应时间(Max Response Time)和内存使用率的百分比(Memory Usage)。在这些性能指标下方，可以查看监控的可视化内容，如客户端请求(Client Requests)和每毫秒的客户端响应时间(Client Response Time，最大值和平均值)。

单击实例(Instances)选项卡来获取 Kibana 实例的详细信息，将看到如下图所示的统计信息：

① 译者注：GC 指 Garbage Collection。



在上图中，可以查看顶部窗格中的信息，例如实例的状态有运行的实例总数(Instance)、请求总数(Requests)、总连接数(Connections)、最大响应时间(Max. Response Time)和内存使用百分比(Memory Usage)。在这些性能指标下方可以查看实例的详细信息，例如实例名称(Name)、实例状态(Status)、实例占用的总内存容量(Memory Size)、平均负载(Load Average)、当前请求数(Requests)和响应时间(Response Time，最大值和平均值)。

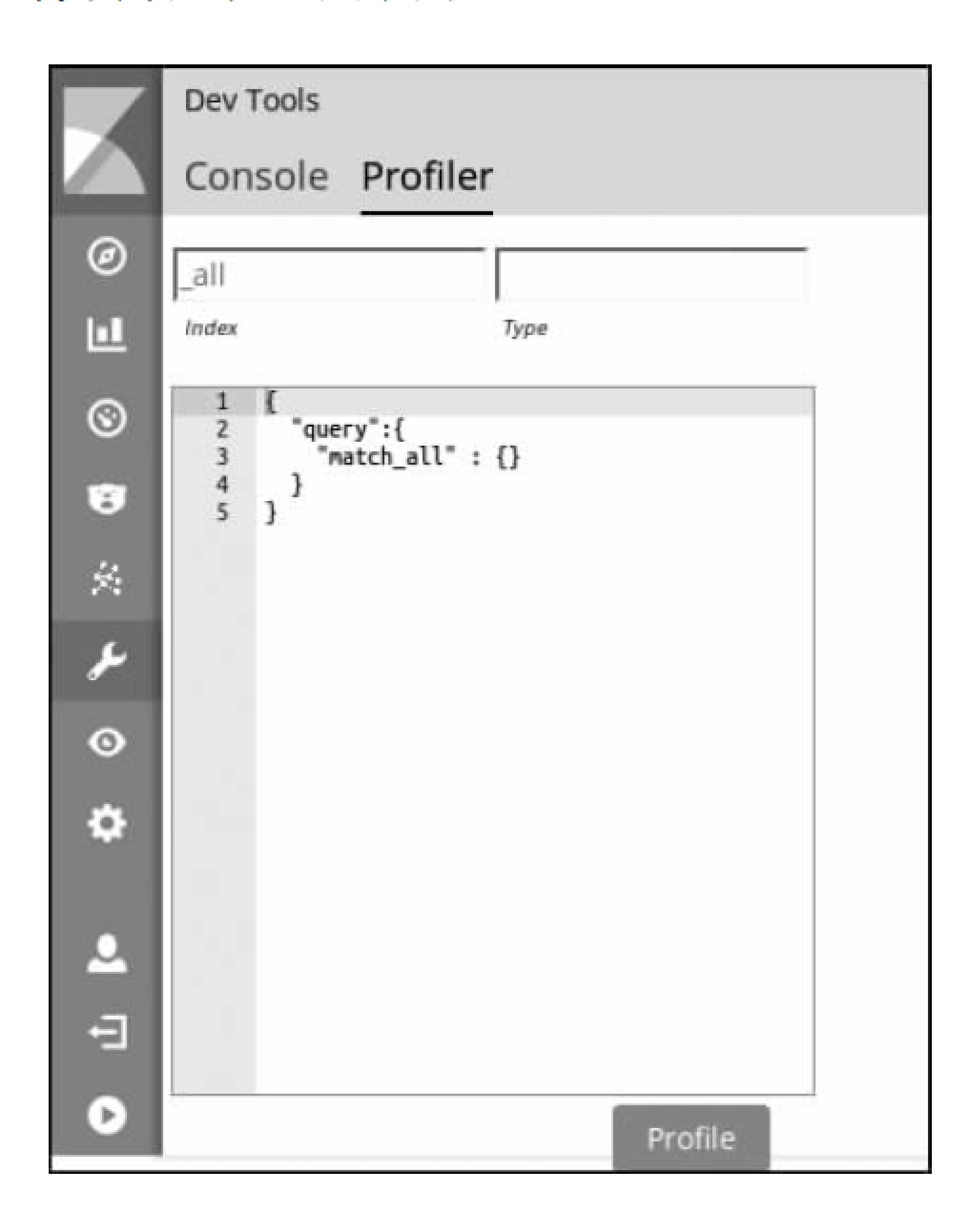
9.6 了解 Profiler

Profiler 是 X-Pack 插件中的另一个组件。这一组件的出现，使我们不再需要了解一条请求在 Elasticsearch 中如何执行。它提供了在各个阶段执行请求的详细信息，这些信息可能有助于找到一些请求执行十分缓慢的原因，或者一些请求返回结果用时过长的原因。

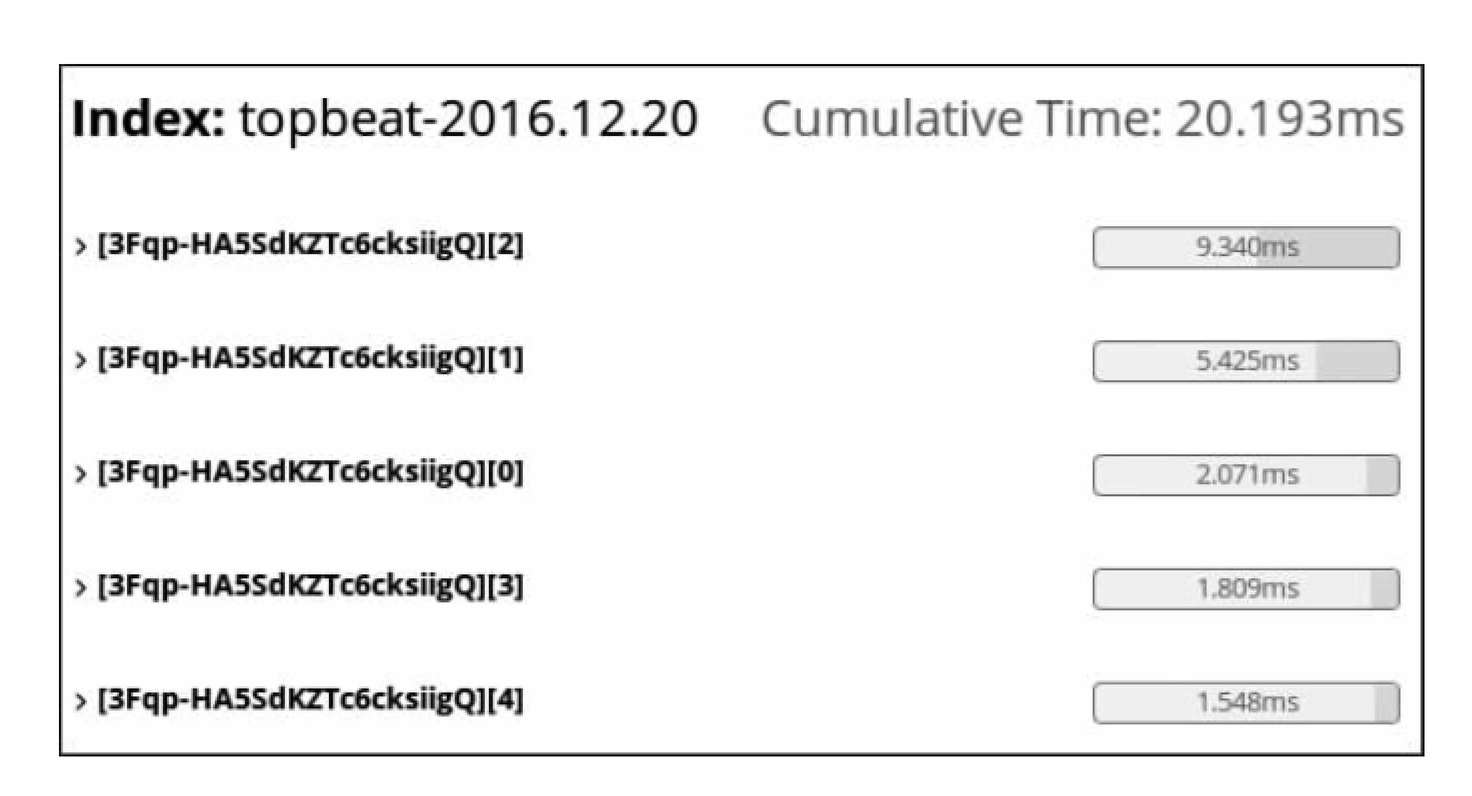
Profile 是 X-Pack 插件最新的附加功能，并已在 X-Pack 5.1 版本中加入。它是 Dev Tools 页面的一部分。它使用由 Elasticsearch 提供的 Profile API。Profile API 用于调试请求中的各种信息，以收集请求的提供方式以及收到的结果等底层详细信息。Profile API 的返回结果信息是大幅的 JSON 数据，这是很难理解的。因此，Profiler 起着重要的作用，可以提供可视化的 Profile API 返回结果，从而使我们更容易理解请求的响应，并调试请求的行为。使用 Profiler，从请求到返回结果，均可了解查询与聚合的中底层信息。

单击 Dev Tools 选项卡，然后单击控制台(Console)右侧的 Profiler 选项卡，可查看

Profiler 的界面。打开后，将会看到以下界面：



如上图所示，可以在请求窗格中指定索引名称(Index)、类型名称(Type)和查询。单击Profile 按钮之后，返回结果将显示在右侧的结果窗格中。将会看到以下结果界面：

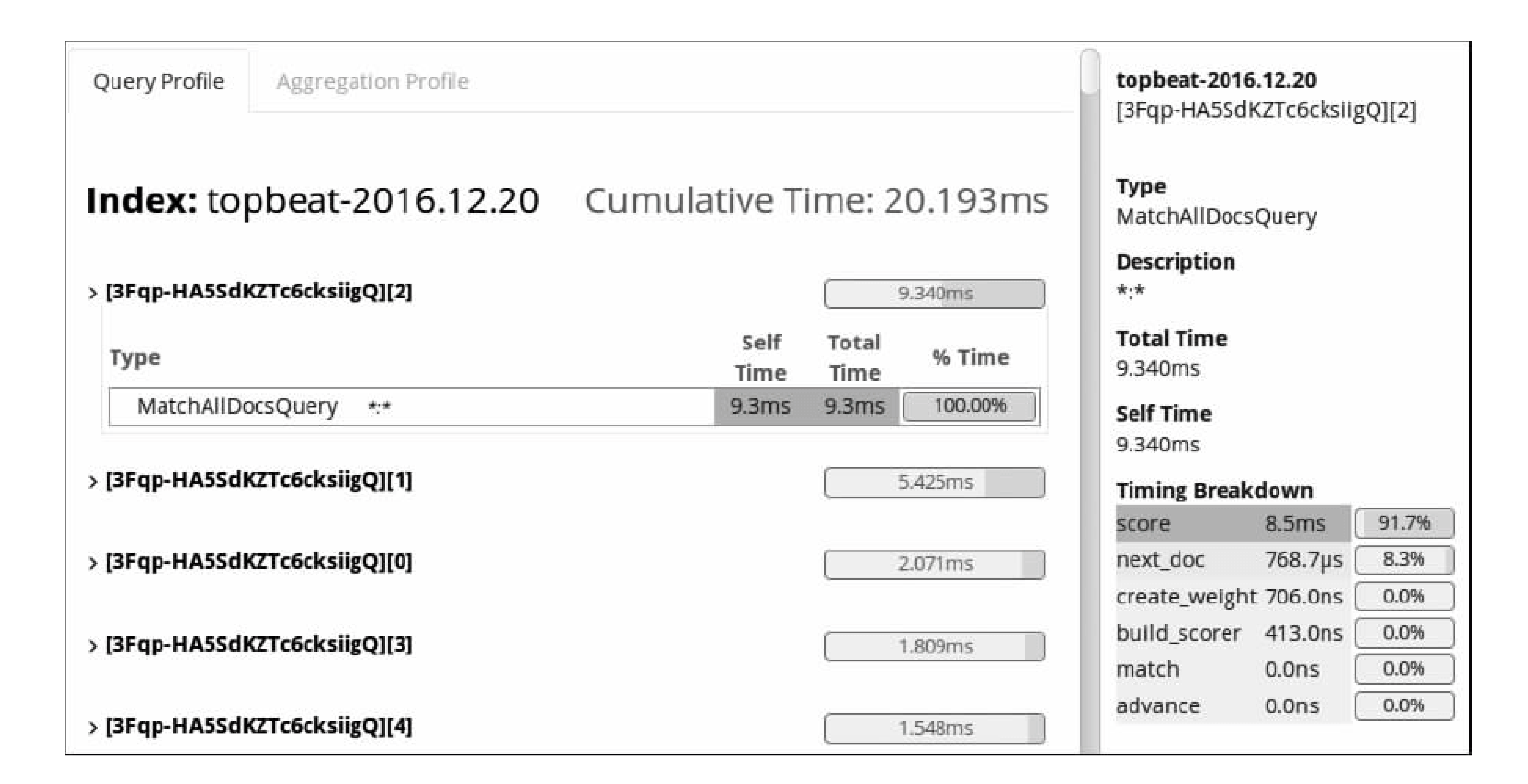


下面解释返回结果的含义。

它显示其搜索过的索引名称、用来在每个索引中搜索的分片以及通过查询获取结果所花费的时间。

> 累积时间是每个时间片的总和，它可以表示通过查询来获取结果所需的实际物理时间。

要查看更详细的信息，可以单击分片名称的左侧，并将光标悬停在类型名称上，该名称将提供更多信息，如下图所示。



它将显示所使用查询的类型，以及 Lucene Query 的各种低级别请求的描述和时间划分。

> 要了解更多关于 Timing 时间划分的组件，可以参考以下链接：
> https://www.elastic.co/guide/en/elasticsearch/reference/5.1/_profiling_queries.html#_timing_breakdown

此外，还可以存储从响应结果生成的 JSON 输出信息来做进一步分析。如果将生成的 JSON 粘贴到 Profiler 请求窗格，它将解析为 Profiler 分析结果而不是查询的结果，并将相应的结果进行可视化展示。

9.7 本章小结

Elastic 团队将 Elastic Stack 中的核心组件和支持功能分别置于 X-Pack 插件中，X-Pack 插件对任何组织而言都是重要组件。在本章中，了解了两个 X-Pack 团队成员：Security 和 Monitoring，掌握了在 Elasticsearch 和 Kibana 中安装 X-Pack 插件的方法。对于 Security 组件，我们学会了管理用户、角色和权限等操作，而 Monitoring 组件则帮助我们查看了 Elasticsearch 和 Kibana 实例实时统计信息的不同可视化展示。

在下一章中，将介绍 X-Pack 插件的其他组件：Alert、Graph 和 Reporting。



X-Pack 插件中的 Alerting、Graph 和 Reporting 组件

安装 X-Pack 插件时，包含六个重要组件，它们全部随插件被安装在系统中。其中三个组件是 Monitoring、Security 和 Profiler，已经在前面的章节中讨论过了。在本章中，将探讨其他组件，了解它们可提供什么功能，并了解它们如何满足我们的需求。

在本章中，将介绍以下部分：

- **Alerting**（警报）和 **Notification**（通知）组件；
- **Graph**（图）组件；
- **Reporting**（报告）组件。

10.1 Alerting 与 Notification 组件

每个组织都需要一个能够在需要时发出警报的系统。X-Pack 插件的这个组件为我们提供了灵活性，可以根据由于数据更改而触发的某些条件来创建警报和通知。由于有实时警报机制这个需求，因此需要这款名为 Watcher 的组件。它可根据指定的条件触发某些操作，在需要时能随时随地创建和执行操作，且不需要在整个 Elastic Stack 中更改配置。如果需要更加细致的监控，可以使用适合的（Logstash）Filter 和 Output 插件，在 Logstash 配置文件中直接指定条件，从而创建这些警报。但是，每次创建新类型的警报时，都需要进行适当的测试才能在生产环境中进行实际的配置。与其在现有设置中进行更改，不如随时创建自己的规则，以获取警报和通知。可以根据需要，跟踪和创建任何类型的警报。

警报（alert）和通知（notification）总是有助于捕获系统出现的任何严重错误或问题。部分错误情况包括但不限于以下内容：

- 监视日志，如果日志包含 **SEVERE**、**FATAL** 或 **ERROR** 的消息，那么可通过电子邮件发送日志消息以通知系统管理员。
- 跟踪各种参数来监视系统框架，如内存使用情况、磁盘使用情况等。创建警报并打

开一个 JIRA 凭证(ticket),可发送内存或磁盘使用率高的通知,此时需要更多可用的磁盘或内存空间。

- 跟踪社交媒体帖子或推文以及指定关键字。当帖子或推文数量超过预定的值时,发送警报。
- 根据用户在 Elasticsearch 中搜索的关键字创建提醒,以便用户可以随时更新。

安装 X-Pack 插件之后,Watcher 组件即可使用,不需要单独安装。然而,目前没有可用的 Watcher 用户界面来创建警报或通知,所有可用的功能都是通过为 Watcher 开放的 API 提供的。

Watcher,如其名称,通过"观察"所有事物、事件和规则来发送通知。使用其 API,可以创建、管理、测试和删除这些 **watch**。每个创建的 watch 可提供单个警报,实际上它可以触发多个通知并发送到多个系统。

让我们从宏观角度看看 Watcher 所需的基本组成部分:

- **Schedule**:调度,定义如何触发 watch。
- **Query**:查询,指定充当起点的查询,根据这样的查询可以指定条件。Watcher 完全支持 Lucene 和 Elasticsearch 查询语言,因此可以指定任何类型的查询。
- **Conditions**:条件,根据查询指定条件。可以根据生成警报的需要来使用 simple 条件或 script 条件。
- **Actions**:操作,在满足条件后执行操作。操作包括向 Slack 通道发送通知,使用电子邮件发送警报等。

简而言之,Scheduler 在正确的时间触发查询。该查询充当条件的输入。对于查询来说,可以指定多个条件。一旦指定了条件,就通过操作来定义在满足条件时要执行的操作。

所有 watch 的完整历史记录都被存储在单独的 Elasticsearch 索引中,用于指定 watch 被触发的时间、watch 的查询、条件是否满足以及基于条件的操作。

对于 Watcher 这样一款作为随 X-Pack 插件安装的组件,可以在控制台中执行以下命令,使用 Watcher API 验证 Watcher 是否已被启用:

```
GET _xpack/watcher/stats
```

执行上述命令将返回如下响应信息:

```
{
    "watcher_state": "started",
    "watch_count": 0,
    "execution_thread_pool": {
        "queue_size": 0,
        "max_size": 0
```

```
    },
    "manually_stopped": false
}
```

返回的信息显示，Watcher 已经启动，但 watch_count 为 0，因为没有定义 watch。

到目前为止，我们已经了解到，可以创建在满足条件时执行操作的 watch。查询搜索和条件检查的数据也称为 **watch 有效负载（Payload）**。

每当指定触发 Schedule 时，它都会将查询结果加载到 watch 有效负载中。一旦数据被加载到 watch 有效负载中，就会创建一个条件。如果条件匹配，则触发进一步的 Watcher 事件。此外，在执行操作之前，可以执行不同类型的变换（可选项），这将会将查询的所有结果数据加载到 watch 有效负载中。可以使用变换来覆盖 watch 有效负载并加载数据，然后将其发送到 Actions。变换可以在 watch 和执行的操作级别中使用。

在学习 Watcher 怎样执行任务，以及在创建 Watcher 时会执行那些步骤之前，先详细了解它的基本框架以及 watch 的结构。

1. 触发器

创建 watch 时，必须指定触发器（trigger），因为它会告诉 watch 何时执行以及多长时间执行一次。每当创建 watch 时，触发引擎都会注册一个触发器，触发引擎会对触发器进行评估，并相应地运行 watch。目前，只有 Schedule 中基于时间的触发器是可用的。

调度触发器（schedule trigger）根据提到的日期或 watch 的触发频率来定义 watch 的执行时间。Watcher 支持各种类型触发器的调度，如时间间隔、cron 表达式、每小时、每日、每周、每月和每年的调度。

（1）时间间隔

它以固定的时间间隔（interval）触发 watch，按照每个指定的间隔触发 watch。可以以秒、分钟、小时、日或周为单位定义时间间隔，默认情况下的单位为秒。

以下面的代码为例：

```
{
    "trigger" : {
    "schedule" : {
        "interval" : "1h"
    }
    }
}
```

上述触发器意味着，一旦创建 watch，每小时将执行一次 watch。同样，可以指定 10 分钟或 10 秒的间隔，这样 watch 将每隔 10 分钟或 10 秒执行一次。这里的时间单位支持小时（h）、分钟（m）、秒（s）、日（d）和周（w）。

（2）cron 表达式

它根据指定的 cron 表达式来触发 watch。cron 表达式为-＜seconds＞＜minutes＞＜hours＞＜day_of_month＞＜month＞＜day_of_week＞[year]。在这里，year 是可选的，其余的字段都是必选的。

cron 表达式的每个字段都有可能取值的集合，如下表所示：

字　段	支持的值	支持的特殊字符
秒	0～59	,-*/
分钟	0～59	,-*/
小时	0～23	,-*/
一个月中的日	1～31	,-*?/LW
月	1～12 或 JAN～DEC	,-*/
一周中的日	1～7 或 SUN～SAT	,-*?/L#
年	空值或 1970～2099	,-*/

> 有关特殊字符的详细信息，请参阅：
> https://www.elastic.co/guide/en/x-pack/5.1/trigger-schedule.html#schedule-cron-elements

以下是对 cron 表达式的解释：

cron 表达式	解　释
0 30 * * * ?	在每小时的第 30 分钟触发一次
0 30 * * * ? 2017	仅在 2017 年中每小时的第 30 分钟触发一次
0 45 1 * * ? 0 30 14 ? * MON	在每天上午 1:45 触发一次 在每个周一下午 2:30 触发一次，每周一次
0 45 17 3 * ?	在每个月第 3 天的下午 5:45 触发一次
0 0 10 27 7 ? *	在每年 7 月 27 日上午 10:00 触发一次

以下面的代码为例：

```
{
    "trigger" : {
        "schedule" : {
            "cron" : "0 30 * * * ?"
        }
```

```
    }
}
```

由于 cron 表达式理解起来有一些难度，为了简化，还有其他调度触发器，可以在每小时、每日、每周、每月和每年的级别轻松地指定时间间隔。

(3) 每小时调度

这种方式在每小时的特定时刻触发 watch。可以在其属性中指定分钟的值，该分钟属性可以每隔一个小时或一小时内指定的分钟的值运行。

以下面的代码为例：

```
{
    "trigger" : {
        "schedule" : {
            "hourly" : { "minute" : 45 }
        }
    }
}
```

执行时，将在每小时的第 45 分钟触发 watch：

```
{ ... "hourly" : { "minute" : [10,20,30,40] }}
```

执行时，将在每小时的第 10、20、30、40 分钟触发 watch。

(4) 每日调度

这种方式每天在固定时间，或者使用 at 属性以及小时和分钟的数组来触发 watch。如果没有指定每日的任何属性，那么默认情况下，它将在午夜运行，即 00:00。每天可以对其完成一次或多次调度。

以下面的代码为例：

```
{
    "trigger" : {
        "schedule" : {
            "daily" : { "at" : "18:30" }
        }
    }
}
```

执行时，每天下午 6:30 将触发 watch：

```
{ ... "daily" : { "at" : ["09:45", "18:30"] }}
```

执行时，将在上午 9:45 和下午 6:30 触发 watch，每天触发两次。

```
{ ... "daily" : { "at" : { "hour" : [ 0,8,22],"minute" : [10,55] }}}
```

执行时,将在上午 12:10、上午 12:55、上午 8:10、上午 8:55、下午 10:10 和下午 10:55 触发 watch。

(5) 每周调度

这种方式在固定时间周期,或使用 on 和 at 等属性触发 watch。

以下面的代码为例:

```
{
    "trigger" : {
        "schedule" : {
            "weekly" : { "on" : "tuesday", "at" : "18:30" }
        }
    }
}
```

执行时,将在每个星期二下午 6:30 触发 watch。

```
{ ... "weekly" : [{ "on" : "tuesday", "at" : "noon" },{ "on" : "thursday", "at" :
"18:30" } ] }
```

执行时,将每周触发两次 watch,即周二下午 12:00 和周四下午 6:30。

还可以指定一个数组的值,如下所示:

```
{ ... "weekly" : { "on" : ["tuesday", "thursday" ],"at" : ["noon", "18:30" ] }}
```

执行时,将在每周二和周四的中午 12:00 和下午 6:30 触发 watch。

(6) 每月调度

这种方式在固定时间,或使用 on 和 at 等属性,每个月触发 watch。

以下面的代码为例:

```
{
    "trigger" : {
        "schedule" : {
            "monthly" : { "on" : "15", "at" : "10:00" }
        }
    }
}
```

执行时,将于每月 15 日的上午 10:00 触发 watch。

```
{ ... "monthly" : [{ "on" : "5", "at" : "midnight" },{ "on" : "25", "at" : "18:30" } ] }
```

执行时,将于每月第 5 天中午 12:00 和第 25 天下午 6:30 两次触发 watch。

还可以指定如下所示的数组:

```
{ ... "monthly" : { "on" : ["5", "25" ],"at" : ["midnight", "18:30" ] }}
```

执行时,将在每个月的第 5 和 25 日的中午 12:00 和下午 6:30 触发 watch。

2. 输入

Watcher 的一个基本功能是将输入数据加载到 watch 有效负载中执行。加载的数据可以有多种不同的格式,例如使用查询或将静态数据加载到 watch 有效负载中等。

Watcher 支持以下输入方式。

1) 简单输入

简单输入(simple input)将静态数据加载到 watch 有效负载中以供执行,从而可存储静态数据,这些数据将成为下一步执行过程的输入。可以将静态数据定义为数值型、字符串或 object 型数据。

以下面的代码为例:

```
"input" : {
"simple" : {
"book" : "Mastering Elastic Stack",
"year" : 2017,
"author_names" : {
    "author1" : "Yuvraj",
    "author2" : "Ravi",
        }
    }
}
```

2) 搜索输入

搜索输入(search input)用于将 Elasticsearch 中搜索的查询结果加载到 watch 有效负载中以执行 Watcher。在这种类型的输入中,可以通过索引名称来指定请求对象,指定索引类型(如果有的话)和搜索请求的正文。搜索输入完全支持 Elasticsearch 的查询语言。

以下面的代码为例:

```
"input" : {
    "search" : {
        "request" : {
            "indices" : [ "logstash- * " ],
            "types" : [ "records" ],
```

```
            "body" : {
            "query" : { "match_all" : {}}
            }
        }
    }
}
```

在这个例子中，指定了包含 request.indices、request.types、request.body 等属性的搜索输入，它们分别用于指定要搜索的索引、类型和文档数据。

还有一些其他属性，如下所示：

- request.search_type：指定要执行搜索的类型，默认值为 query_then_fetch。
- request.template：指定搜索模板的正文，可以包含静态或动态的数据。
- extract：从搜索请求的结果中提取特定字段，并将其作为有效负载来加载。如果搜索查询提供了很长的响应，则可以使用 extract 来选择必需的字段。
- timeout：指定搜索查询响应的时间限制，默认值为 30 秒。
- request.indices_options.expand_wildcards：将索引名称中包含的通配符扩展为带有 all、none、open 或 closed 等值的数据。
- request.indices_options.ignore_unavailable：指定是否忽略当前不可用的索引，其值为 true 或 false。
- request.indices_options.allow_no_indices：指定在找不到索引时是否允许搜索，其值为 true 或 false。

搜索输入是 Watcher 中最常见的输入形式，可于定义条件、变换(transform)或操作。为了了解如何使用搜索的结果进行进一步评估，我们来看看访问搜索结果的各种方法。

了解如何访问搜索结果十分重要。对搜索结果的访问可以用于处理 watch 有效负载以执行后续步骤，如条件、变换或操作。

- 通过指定 ctx.payload.hits 加载所有搜索结果。
- 仅使用 ctx.payload.hits.total 加载搜索结果的总数。
- 使用从零开始的数组的索引来加载特定搜索结果中的值。要访问第四个命中的值，请使用 ctx.payload.hits.hits.3。
- 使用 ctx.payload.hits.hits.<index> .fields.<fieldname>从搜索结果加载特定字段的值，可以在其中指定从零开始的数组的索引，并指定字段名来提取字段值。

3) HTTP 协议输入

HTTP 协议输入(HTTP input)使用 HTTP 服务终端来加载搜索请求的结果，并保存到 watch 中。可以使用 HTTP 输入与任何开启 HTTP 监听的 Web 服务进行交互，例如查询第三方服务、查询外部 Elasticsearch 群集，或查询 Elasticsearch API。

还有一些其他属性，如下所示：

```
"input" : {
    "http" : {
        "request" : {
            "host" : "localhost ",
            "port" : 9200,
            "path" : "/_search",
            "body" : "{"query" : { "match" : { "loglevel" : "error"}}}"
        }
    }
}
```

调用 Elasticsearch API 的示例如以下代码所示：

```
"input" : {
    "http" : {
        "request" : {
            "host" : "localhost ",
            "port" : "9200",
            "path" : "/_nodes/stats",
            "params" : {
                "human" : "true"
            }
        }
    }
}
```

params 将以适合阅读的格式返回。

要了解如何使用 HTTP 响应的结果做进一步评估，我们来看看访问搜索结果的各种方法。

了解如何访问 HTTP 响应结果非常重要，可以将其用于处理 watch 有效负载，以便执行后续的步骤，如条件、变换或操作。如果 HTTP 的响应采用 JSON 或 YAML 格式，则响应将加载到 watch 有效负载中，否则将加载到 watch 有效负载中的_value 字段中：

- 通过指定 ctx. payload. message 加载所有消息数据。
- 通过使用 ctx. payload. _headers 从响应中加载所有响应头。
- 通过指定 ctx. payload. _status_code 加载所有 HTTP 状态码。

4) 链式输入

链式输入(chain input)用于加载来自多个源的数据或结果，或组合多种类型的输入，如简单、搜索或 HTTP 协议等输入。它按指定的顺序进行处理。可以将输入获取到的数据无

缝地送入另一个输入中，然后将此数据由 Watcher 的任何一部分执行操作。这样，第一个输入的输出可以被第二个输入使用，以此类推。

以下面的代码为例：

```
"input" : {
    "chain" : {
        "inputs" : [
            {
                "first" : {
                    "simple" : { "field" : "loglevel" }
                }
            },
            {
                "second" : {
                    "search" : {
                        "request" : {
                            "indices" : [ "logstash"],
                            "body": "{"query": { "match":
                                { "{{ctx.payload.first.field}}":"error"}}}"
                        }
                    }
                }
            }
        ]
    }
}
```

3. 条件

了解 Watcher 如何工作是很重要的。将数据加载到 watch 有效负载后，利用条件 condition 确定要捕获的内容以便发送警报或通知。当符合条件时，将触发相应的操作。

Watcher 支持以下条件。

1）总是(always)条件

该条件始终将 watch 条件设置为 true，即始终执行 watch 操作，除非正在使用基于时间的 throttling① 来限制操作的执行。如果不指定任何条件，默认情况下为 always 条件。在 always 条件中，没有属性可以指定。

① 可以通过对操作施加一定的限制，使一些始终执行的操作因受限而有选择性地执行，这样的调节称为 throttling，详见后面的 5. 操作。

以下面的代码为例：

```
"condition" : {
    "always" : {}
}
```

2）从不（never）条件

该条件始终将 watch 状态设置为 false，任凭动作被触发，watch 的操作将永远不会被执行。这种情况主要用于测试 watch 是否按要求工作，并将其录入到 watch 历史记录中。在 never[①] 条件中，没有属性可以指定。

以下面的代码为例：

```
"condition" : {
    "never" : {}
}
```

3）比较（compare）条件

比较（compare）条件对输入获得的结果进行基本比较。它可比较条件与 watch 有效负载 Payload 中所接收的结果。

支持的比较运算符如下：

- eq：如果结果值与给定值匹配，则返回 true（对数字、字符串、列表、对象和值有效）。
- not_eq：如果结果值与给定值不匹配，则返回 true（对数字、字符串、列表、对象和值有效）。
- gt：如果结果值大于给定值，则返回 true（在数字和字符串上有效）。
- gte：如果结果值大于或等于给定值，则返回 true（在数字和字符串上有效）。
- lt：如果结果值小于给定值，则返回 true（在数字和字符串上有效）。
- lte：如果结果值小于或等于给定值，则返回 true（在数字和字符串上有效）。

以下面的代码为例：

```
{
    "condition" : {
        "compare" : {
            "ctx.payload.hits.total" : {
                "lt" : 10
            }
        }
```

① 译者注：此处的原文是“always condition”，通过查看上下文，笔者认为此处存在错误，将其视为“never condition”进行翻译。

```
        }
    }
```

在上面的例子中，执行的是比较已经加载到 watch 有效负载中的一个字段的结果，以及指定加载的结果中的条件，找到存储在有效负载中的记录的总计数，并判断该计数是否小于10。如果小于 10，则条件将为 true，操作将被触发。

此外，还可以使用日期-数学表达式，甚至比较两个字段的值。

日期-数学表达式的示例如下所示：

```
{
    "condition" : {
        "compare" : {
            "ctx.payload.hits.hits.2.fields.datetime" : {
                "gte" : "<{now-1h}>"
            }
        }
    }
}
```

在上面的例子中，执行的是比较已经加载到 watch 有效负载中的一个字段的结果，以及指定加载结果中的条件，查找第三条记录，并检查过去一小时内发生的字段(datetime)的值。

4）数组(array)比较条件

数组(array)比较条件用于在输入获得的结果中对数组进行基本比较。它的基本结构包括数组、字段的路径(可选)、比较运算符以及与之比较的值。

以下面的代码为例：

```
{
    "condition": {
        "array_compare": {
            "ctx.payload.aggregations.twitter.buckets" : {
                "path": "doc_count",
                "gte": {
                    "value": 1000,
                }
            }
        }
    }
}
```

在上面的例子中，已经指定了数组，并且还指定了一个 bucket 的路径。当 bucket 的文档数量大于或等于 1000 的时候，条件设置为 true。

5）脚本（script）条件

脚本（script）条件用于验证基于脚本的 watch 条件，其中的脚本决定了操作是否执行。可以使用 Elasticsearch 支持的脚本语言，默认使用的脚本语言是 **Groovy**。

> 要使用 Groovy 作为脚本语言，需要在 elasticsearch.yml 中启用动态脚本。要启用 Groovy 中的 watch，请设置以下属性：
>
> ```
> script.engine.groovy.inline.xpack_watch: true
> ```
>
> 此外，随着 Elastic Stack 的推出，出现了一种名为 **Painless** 的新型基本脚本语言，这种脚本的使用无须启用动态脚本。

脚本条件的一个例子如下：

```
"condition" : {
    "script" : "return false"
}
```

这是一个简单的条件，总是返回 false。

与脚本条件相关联的几个属性有：

- **脚本类型（script type）**：指定要使用的脚本类型（支持的脚本：inline、file 和 stored）
- **脚本语言（scripting language）**：指定要使用的脚本语言（支持的语言：Groovy、JavaScript、Python、Painless、Expression、Mustache 和 Java）。
- **参数值（parameter values）**：指定参数（如果有的话）。

（1）Inline 脚本

在这种类型的脚本中，可以在条件本身内部直接定义脚本。

以下面的代码为例：

```
"condition" : {
    "script" : {
        "inline" : "return ctx.payload.hits.total >threshold",
        "lang" : "painless",
        "params" : {
            "threshold" : 5
        }
    }
}
```

在上例中，使用脚本条件来检查总结果数是否大于定义的阈值参数。

(2) file 脚本

在这种类型的脚本中，可以定义文件中存在的脚本。脚本应存储在 $ES_HOME/config/scripts 目录中。

以下面的代码为例：

```
"condition" : {
    "script" : {
        "file" : "watcher_script",
        "lang" : "python",
    }
}
```

在上面的例子中，参考 watcher_script.py(watcher_script 是文件的名称，其后的.py 是 Python 语言的扩展名)文件来检查条件。

(3) Stored 脚本

这种类型的脚本指的是已经存储在 Elasticsearch 中的脚本。以下面的代码为例：

```
"condition" : {
    "script" : {
        "id" : "test_script ",
    }
}
```

在本章的前面部分，已经看到了如何访问一个搜索/ HTTP 返回的结果。在 Watcher 上下文中，可以访问更多变量，如下所示：

- ctx.watch_id：访问正在执行的 watch 的 ID。
- ctx.execution_time：watch 启动的执行时间。
- ctx.trigger.triggered_time：watch 触发的时间。
- ctx.trigger.scheduled_time：watch 预计触发的预定时间。
- ctx.metadata.*：如果关联，则访问元数据记录。
- ctx.payload.*：访问 watch 加载的有效负载数据。

4. 变换

变换(transform)是一个可选的项目，可以在创建 watch 时进行定义。其目的是在触发动作之前更新由 watch 有效负载所加载的数据。在触发了使用输入和条件将数据加载到有效负载的 watch 之后，和在触发操作之前，我们可以使用变换来更改 watch 有效负载中的数据。由于变换是可选的，如果不定义它们，则加载到 watch 有效负载中的数据将被用于执行

操作。变换可以定义为顶级字段或用于操作中。如果用到的变换将要使用相同的有效负载 payload，则使用变换作为顶级字段。如果变换需要不同的有效负载 payload 的结果，则应在操作定义中使用变换。

下面介绍一些 Watcher 支持的变换。

1）搜索变换

在这种变换中，将执行搜索查询，搜索查询的结果将替换当前 watch 有效负载中的数据。其使用方法与搜索输入及其属性相同。

以下面的代码为例：

```
"transform" : {
    "search" : {
        "request" : {
            "indices" : [ "logstash- * " ],
            "types" : [ "records" ],
            "body" : {
                "query" : { "match_all" : {}}
            }
        }
    }
}
```

2）脚本变换

在这种变换中，它将执行脚本，脚本执行的结果将替换当前 watch 有效负载中的数据。它的使用方法与脚本输入及其属性相同。

以下面的代码为例：

```
{
    "transform" : {
        "script" : "return [ time : ctx.watch_id ]"
    }
}
```

3）链式变换

在这种变换中，它将按照已指定的顺序执行变换，并且所应用变换的结果将替换当前 watch 有效负载中的数据。它用于构建包含搜索和脚本转换的复杂变换。其使用方法与链式输入及其属性相同。

以下面的代码为例：

```
"transform" : {
```

```
"chain" : [
{
    "search" : {
        "request" : {
            "body" : { "query" : { "match_all" : {} }}
        }
    }
},
{
            "script" : "return [ doc_count: ctx.payload.hits.total ]"
        }
    ]
}
```

5. 操作

操作(action)是创建 watch 时定义的最后一个重要的部分。它用于对数据执行操作,例如记录数据,将数据存入索引,以及向 e-mail、hipchat 或 slack 发送警报和通知等。只有在满足 watch 的条件时才会执行操作。单个 watch 可以包含多个动作,动作中甚至可以包含变换。所有动作按指定顺序逐个执行。如果执行动作失败,则相关的信息会存储在 watch 的历史记录中。操作在触发器中不是必须指定的。

在操作中,有一个 throttling 的概念,下面我们来了解它的用法。

throttling 用于限制 watch 多种操作的执行。为了简单地理解它,假设创建了一个 watch,可以在获得的推文中找到 watcher 一词的数量。watch 每两分钟触发一次,并在过去 30 分钟内搜索“watcher”。当 watch 触发并发现“watcher”的时候,若条件为真,它就会每隔两分钟重复发送相同的警报,这将多次触发相同的操作。

为了解决这些问题,Watcher 支持 throttling:基于时间和确认的 throttling。

1) 基于时间的 throttling

在这种类型的 throttling 中,可以定义动作中的 throttling 周期,然后将始终检查在 throttling 时间段(当前时间:throttling 周期)中是否已执行动作。当一个操作将被执行时,由于有时受到 throttling 的限制,该操作可能最终不会执行。因此,Watcher 不会每隔两分钟发送警报,而是依照 throttling 的设置每隔 15 分钟发送警报。可以使用操作定义中的 throttle_period 关键字来定义这个时间间隔,也可以为不同的动作设置不同的 throttling 时间。

此外,也可以在 watch 的级别定义 throttle_period,这将确保按照 throttling 设置的时段来执行操作。在这种情况下,不同的操作将拥有各自独立的 throttle_period。不能为不同的操作设置不同的 throttling 次数。

> ℹ 如果没有定义 throttle_period，则默认设置为 5 秒。可以在 elasticsearch.yml 中添加以下属性来更改 throttle_period 的默认值：
>
> ```
> xpack.watcher.execution.default_throttle_period: 5m
> ```

2）基于确认的 throttling

在这种 throttling 中，可指定条件来多次限制操作的执行。一旦条件被判断为 true，那么对应的操作将始终在 watch 运行的时期执行，或按照 throttle_period 的设置来执行操作。为了限制这种操作，基于确认的 throttling 应运而生。当条件在一开始设置为 true 时，将触发操作一次；如果条件在此之后保持为 true，则不会再触发操作。然后，当条件变为 false 时，它将再次触发操作。当 watch 的动作被确认时，其状态改变为指定的状态；当条件判断为 false 时，其状态变为 awaits_successful_execution。

要确认一个 watch，应使用 Ack Watch API。可以像下面的命令那样使用 API 来确认操作：

```
POST _xpack/watcher/watch/<id>/_ack/
```

在这里，<id>是 watch 的 ID，它会确认所有的操作。

要确认具体操作，请执行以下命令：

```
POST _xpack/watcher/watch/<id>/_ack/<action_ids>
```

Watcher 支持以下操作：

- **E-mail**：通过电子邮件发送警报或通知。可以包含纯文本或 HTML 格式的文本，也可以在发送电子邮件时添加附件。Watcher 可以将邮件发送到任何支持 SMTP 服务的电子邮箱中，利用 Watcher 发送邮件需要在 elasticsearch.yml 文件中配置电子邮件。
- **HipChat**：向 HipChat 用户或房间发送警报或通知。需要在 elasticsearch.yml 文件中配置 HipChat 账户，以便使用 HipChat 发送消息。
- **Index**：为 Elasticsearch 中的新数据创建索引，可以自定义警报存储索引的类型名称。在 Index 中使用 transform 时，对_doc 有效负载字段有一个关键的支持。当_doc 字段存在时，将会检查它是否包含 object 或数组类型的对象。如果它包含一个对象，将作为单个文档创建索引；如果存在一个对象数组，则将每个对象视为一个文档。
- **JIRA**：直接利用操作在 JIRA 中创建问题。它需要在 elasticsearch.yml 文件中配置 JIRA 账户以创建 JIRA 问题。
- **Logging**：将数据记录到 Elasticsearch 日志中。它主要用于调试或将警报添加到日

志中，以便被管理员获知。

- **PagerDuty**：在 PagerDuty 中创建事件以进行正确的事件解析。它需要在 elasticsearch.yml 文件中配置 PagerDuty 账户，以创建 PagerDuty 事件。
- **Slack**：向 Slack 用户或团队频道发送警报或通知。它需要在 elasticsearch.yml 文件中配置 Slack 账户，以便使用 Slack 发送消息。
- **Webhook**：向 Web 服务发送警报或通知。支持 HTTP 和 HTTPS 的连接，支持在 elasticsearch.yml 配置文件中定义用户名和密码。用户名和密码以纯文本形式存储在 watches 索引中，可以启用 X-Pack 安全性，Watcher 将对密码进行加密并存储。还可以使用基于 PKI 的身份验证。

我们来看几个使用操作的例子。Index Action 的示例如下所示：

```
"actions" : {
    "index_payload" : {
        "transform": { ... },
        "index" : {
            "index" : "watcher",
            "doc_type" : "watches",
            "execution_time_field " : "watch_executed_time",
            "timeout" : "30s"
        }
    }
}
```

在上面的例子中，index_payload 是操作的 ID，transform 是可选的。在索引操作中，将索引名称定义为 watcher，文档类型名称定义为 watches，其中将会存入数据。字段 index 和 doc_type 都是必须指定的。这里有两个可选字段，即 execute_time_field——用于指定 watch 存储执行时间的字段；以及 timeout——用于指定因在规定时间内没有收到响应而导致 API 调用失败的时间间隔。默认情况下，超时时间为 60 秒。

以下是日志记录(logging)操作的示例：

```
"actions" : {
    "log" : {
        "transform" : { ... },
        "logging" :{
            "text" : "triggered at {{ ctx.trigger.triggered_time}}",
            "level" : "info"
        }
    }
}
```

在上面的例子中，log是操作的ID，transform是可选的。在日志记录操作中，定义要记录日志并存入Elasticsearch的文本。文本字段是必须指定的。有两个可选字段，即用于指定要记录文本的类别的category字段，以及用于指定日志中文本日志记录级别的level字段。日志级别中可能用到的值有error、warn、info、debug和trace。

> 要了解Watcher的完整示例，可以参考第12章**案例分析——Meetup**中的**获取通知**部分。

10.2 Graph组件

Graph组件是X-Pack插件中的另一个附加功能，早期作为独立组件来使用，也称为Graph。它只是将完整的Graph应用程序添加到X-Pack中，而不需要单独安装。这是一个非常强大的工具，旨在发现词项之间如何相互关联。它提供了一种简单而优雅的方式来发现索引中词项之间的关联以及它们的意义。

Graph组件可以在广泛的应用程序（如推荐引擎）中发现多个领域的信息以及它们是如何相互关联的；展示如何在社交网络上与朋友建立联系；进行网站分析，包括漏洞跟踪、访问最多的页面、在页面中向用户推荐等。

可以从两个来源使用Graph，即使用Elasticsearch提供中的Graph Exploration API，或使用Kibana中的交互式Graph可视化工具。Graph的最大优点是只需要对它进行安装，它不会对Elasticsearch索引进行任何更改，可以直接发现数据关联而不会存储任何其他数据。关于这一点是如何做到的，将在下面进行介绍。

Graph直接出自图论，它创建了一个链接型结构来展示多个对象之间的关系。Graph可以被看作是发现对象之间关系的大型网络。在图论中，有节点（node）和边（edge）。在本书的内容中，有顶点（vertice）和连接（connection）。顶点表示多个术语之间的联系。连接表示两个顶点之间的关系，以及它们如何连接，并且总结了两个顶点中的文档信息。

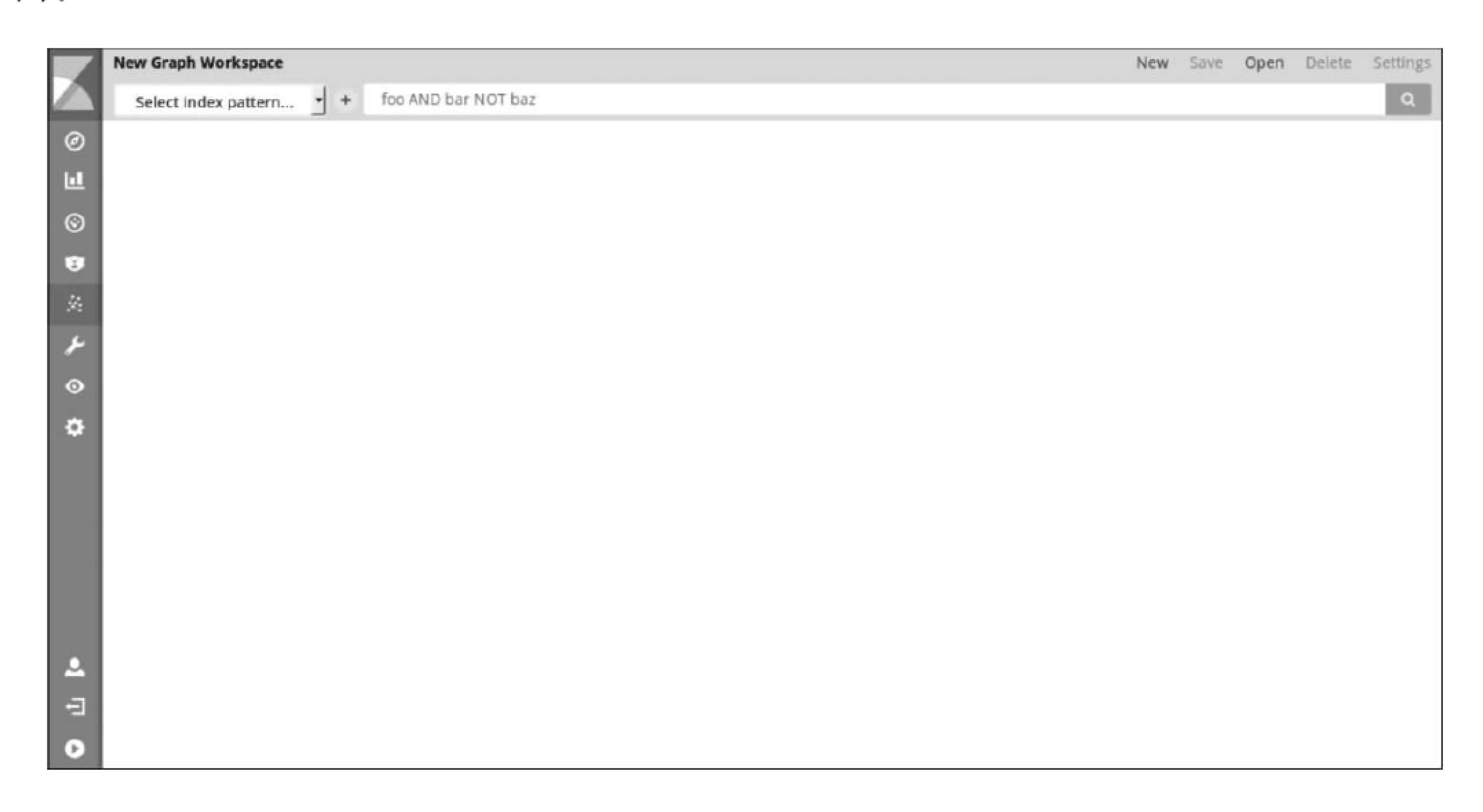
在Elasticsearch中，Graph能够做到开箱即用，其中的顶点引用索引中的词项，其连接利用强大的Elasticsearch聚合来实时创建。此外，使用Elasticsearch的scoring功能，可以发现连接及其相关性。Graph组件能够获取解决困难问题的方案，获取的过程需要多个连接。Graph使用了Elasticsearch聚合框架，能够快速总结数百万个文档。它能够获取使用文档计数的词项间的关系和连接。Graph使用多值字段或多个单值的字段，并在其上创建连接。

在一个简单的场景中，有两个顶点（词项）和它们之间的连接。单击连接后，可了解词项

1 出现的文档数量、词项 2 出现的文档数量，以及同时包含词项 1 和词项 2 的文档数量。这种总结用于建立多个顶点之间的关系。

下面将了解在 Kibana 中，作为 X-Pack 组件之一的 Graph 的用户界面。

要查看 Graph UI，可以单击 Kibana 左侧窗格中的 **Graph** 选项卡。单击后，将看到以下界面：



在上图中，可以看到这个界面十分简单，其中有一个包含各种选项的下拉列表、带有搜索栏的字段添加按钮，以及工具栏中的选项/设置。第一次看到这个界面时，会感觉它很容易使用。没错，你的感觉很对。下拉列表用于选择索引名称或索引模式。下面的加号（+）用于选择要获取连接的字段。搜索栏用于输入查询的词项。工具栏中的操作依次为新建、保存、打开、删除和设置。

让我们借助一个例子来更清晰地了解 Graph UI。创建一个包含多字段的虚拟电子商务订单数据索引。下面要找出这些国家和城市最流行使用哪种支付方式，以及这样的网络中相互连接的人名之间存在何种关系。

要加载示例数据，请执行以下命令：

```
bin/logstash -f sales.conf
```

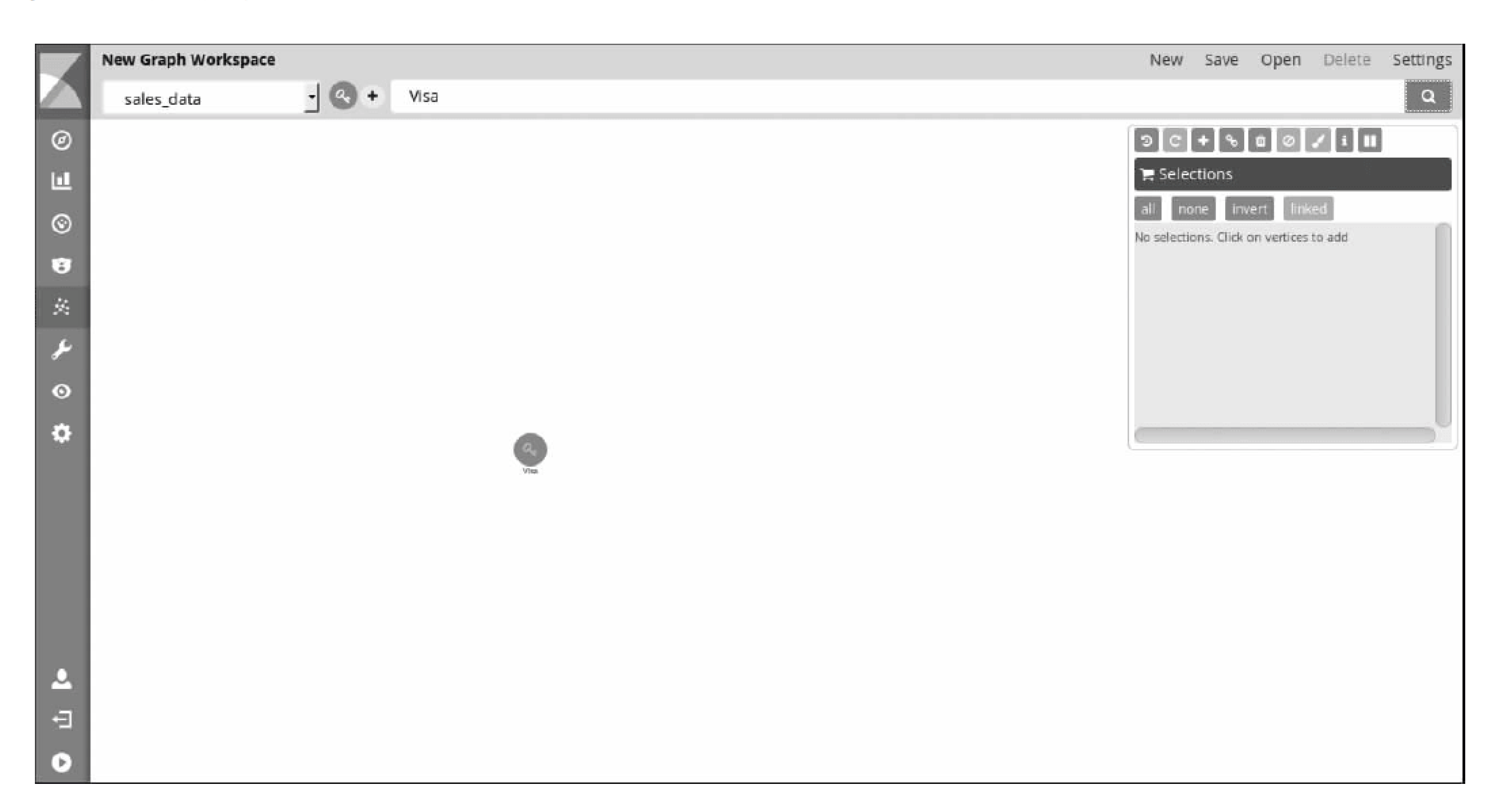
命令中指定的 sales.conf 文件包含 Elasticsearch 输出中的两个附加参数，即用于连接 Elasticsearch 集群的用户和密码。还可以额外创建用 Logstash 发送数据的用户和角色。Logstash 的特定角色将需要 manage_index_templates 和群集监视特权，以及对 Logstash 索引的 write、delete 和 create_index 特权。默认情况下，可以使用 elastic 账户登录。

> ❶可以在如下 Github 页面上找到上述配置文件：
>
> https://github.com/kravigupta/mastering-elastic-stack-code-files/tree/5.1.1/Chapter10

将数据加载到 Elasticsearch 后，转到 Kibana UI 中的 **Management** 选项卡，然后单击 **Index Pattern**，将索引名称添加为 sales_data，选择@timestamp 作为时间字段。

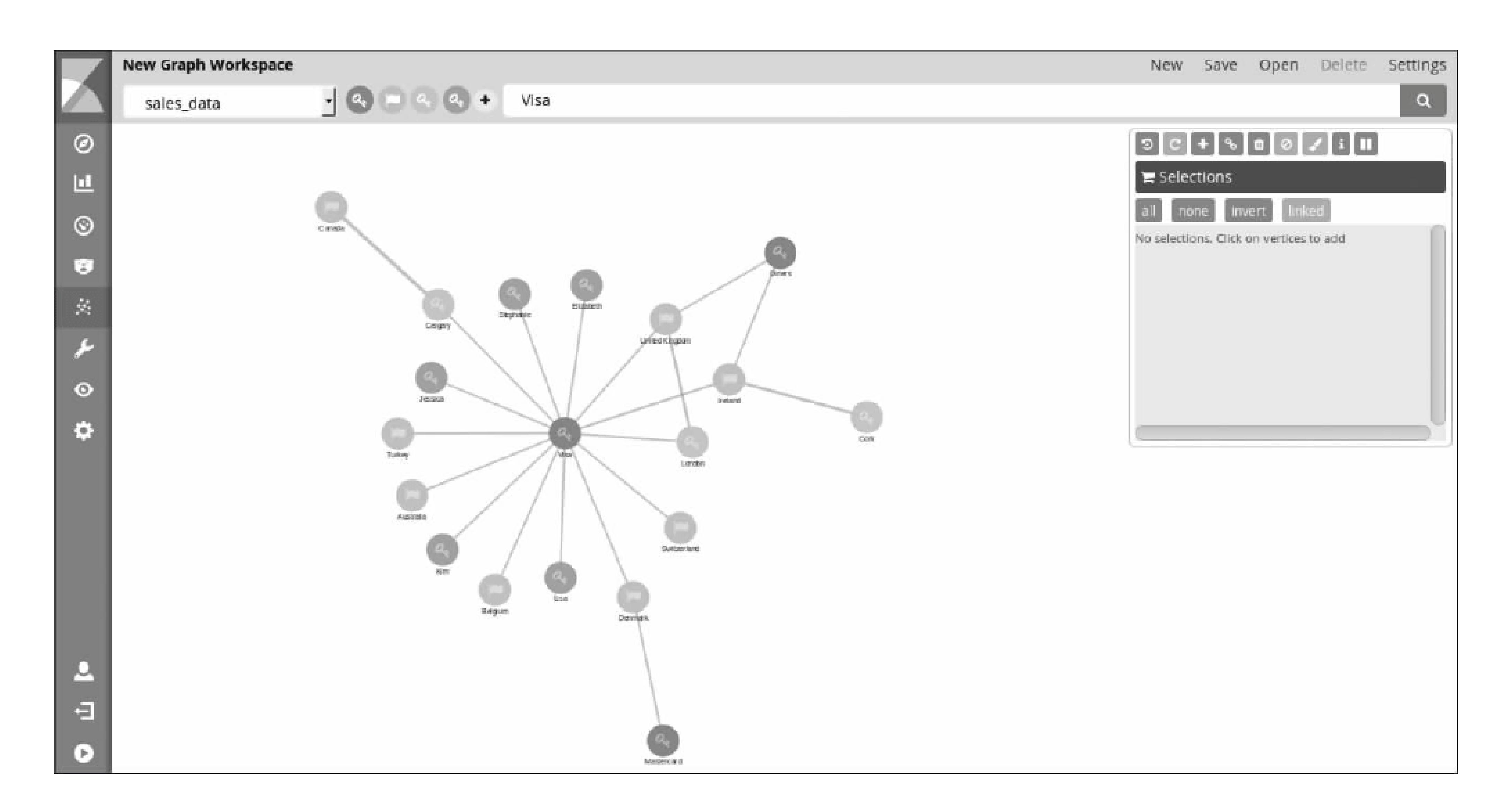
> ❶这些用到的数据源自一个公开的数据集，可以访问如下链接获取：
>
> https://support.spatialkey.com/spatialkey-sample-csv-data/

在将索引名称指定为 sales_data 时，单击加号（+）将字段名称作为 payment_type.keyword，并搜索 Visa，得到如下结果：



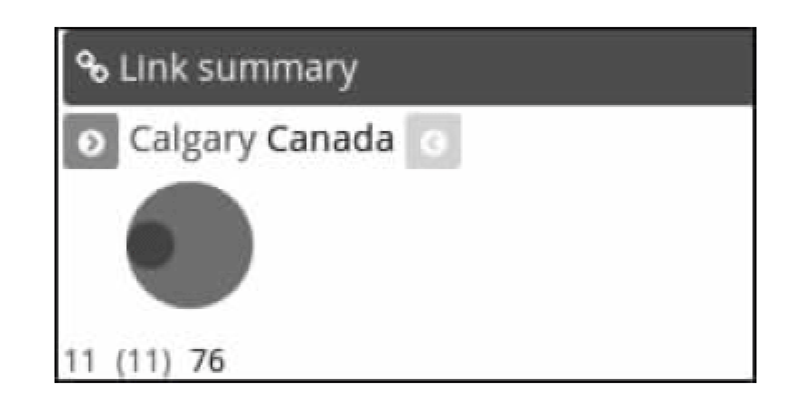
在上图中可以看到，没有给出任何有意义的信息，因为指定单个字段，这样的搜索结果不会显示多个字段或值之间的任何关联。要查看数据间的关系，应在选择 payment_type.keyword 的同时逐个选择多个字段，如 country.keyword、city.keyword 和 name.keyword。选择一个字段后，会看到字段的一些选项，例如颜色、字段图标和每词查询显示结果的最大数量，也就是每个搜索字段将返回的最大数量。

在搜索栏中，指定 Visa，希望看到使用 Visa 付款的人与其所在国家和城市之间的关联。单击**搜索**(**Search**)后，将获得以下图表：



在上图中，可以看到各种连接。两个顶点之间的连接越强，其连接线越暗，如 **Calgary** 到 **Canada** 的连接线。可单击任何一条连接线来了解两个顶点是如何相互关联的。

例如，单击 **Calgary** 和 **Canada** 之间的连接线，观察到有 76 个文档包含词项 **Canada**，11 个文档包含词项 **Calgary**，11 个文档同时包含 **Canada** 和 **Calgary** 两个词项。图中描绘了它们的关系，如下图所示：



在继续操作之前，让我们详细了解 **Selections** 对话框。这个对话框如下图所示：



Selections 对话框包含两组选项，即设置（settings）和选项（selections）。

设置中提供了多种选项，用于指定相关的设置项，从左到右依次为撤销、重做、展开选项、在现有词项之间添加连接、从工作区（workspace）中删除顶点、黑名单选项、自定义选定的顶点样式、向下钻取（获取更深层数据）和暂停布局。

执行从工作区中删除顶点的操作，将删除工作区中不需要的顶点。只需选择不需要的顶点，然后单击删除（看起来像垃圾/回收站）按钮。

可以将选项列入黑名单，这样在对搜索关键字进行更改或其他操作之后，顶点将无法再回到工作区。为此，选择顶点并单击 blocking（看起来像对操作的拦截）按钮。

自定义选定顶点的样式用于更改顶点所在圆形的颜色。

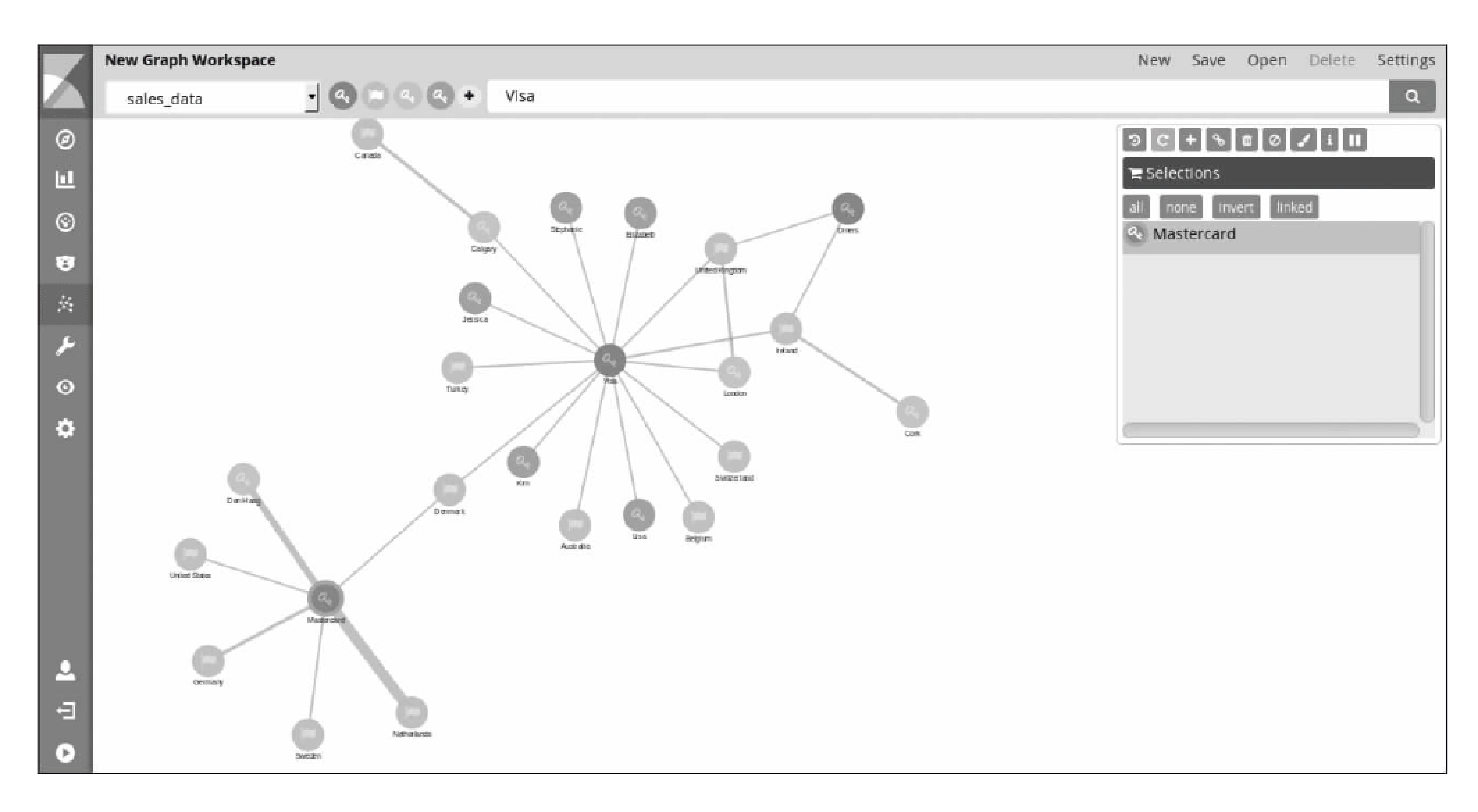
向下钻取用于查看可视化背后的原始文档。

> 如果使用 sales_data 的样本数据，请确保时间过滤器设置为**过去 10 年**或与之等效的时间设置，设置的时间段必须包含 2009 年。

选项中提供了四种选择方式，即 **all**、**none**、**invert** 或 **linked**。其中 all 用于选择所有词项，none 用于选择零个词项，invert 是选择当前未选择的所有词项，linked 是仅选择通过连接彼此相连的词项。

单击任意一个词项之后，该词项将显示在 **Selection** 对话框中。单击 **United Kindom**，可以设置更改 **United Kindom** 的显示名称。把它重命名为 UK，此时它在 Graph 中的显示名称也随之相应改变。

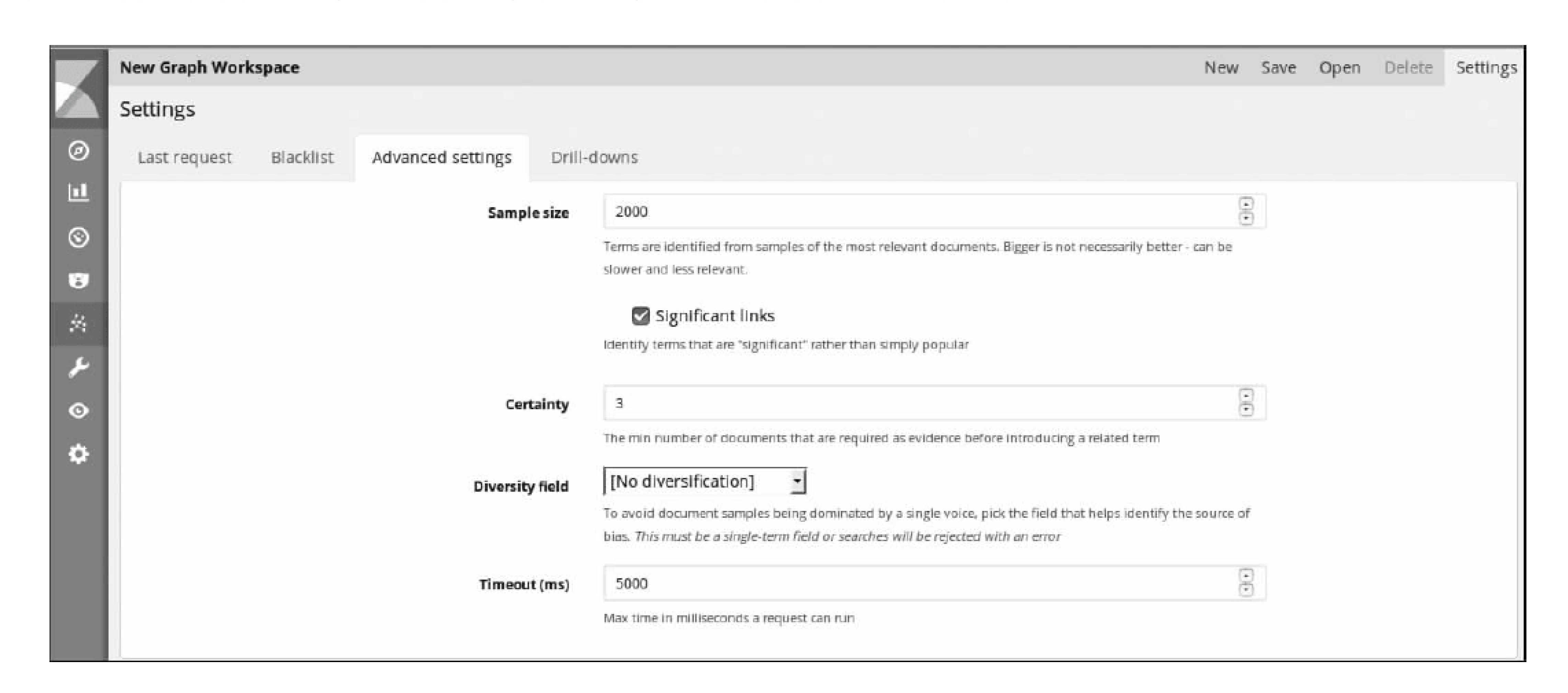
在 Graph 中，可看到与 Mastercard 相关的联系，就像看到 Visa 一样。要做到这一点，只需选择 Mastercard 顶点，然后单击扩展选择符号，将展开其结果，并显示下图：



> ℹ由于规定了每次查询结果的最大数量为5，所以选择Mastercard后，得到了5个结果。

在Graph工具栏中有几个选项，例如创建新工作区、保存工作区、打开已保存的工作区、删除当前保存的工作区以及设置选项。

在设置选项中，可以看到四个子选项，包括最近的请求（Last Request）、黑名单（Blacklist）、高级设置（Advanced Settings）和向下钻取记录（Drill-downs）。单击设置选项，默认情况下会打开高级设置，其中提供了以下设置，如下图所示。



在**最近的请求**选项中，可看到发送到Elasticsearch来获得数据关联的最后一个请求。

在**黑名单**选项中，可看到已经列入黑名单的词项在当前Graph工作区中的重现。

在**向下钻取**选项中，可以指定源代码以及获取在单击向下钻取选项时显示原始文档的URL。可以指定或更改向下钻取选项的标题和工具栏图标。

10.3 Reporting组件

Reporting组件是Kibana最后一个重要的附加功能。它提供了最受欢迎的报告功能，其中包含可用于外部的面板、可视化内容和搜索结果，或者在演示文稿中添加报告等。Reporting使用PhantomJS从图像创建报告并以PDF的形式适当拼接，带来了这项期待已久的功能。

要生成报告，可在**Discover**、**Visualize**和**Dashboard**页面中的工具栏中选择**Reporting**选项。单击这一选项，然后单击**生成可打印的PDF（Generate Printable PDF）**按钮或单击**生成URL（Generate URL）**按钮。

我们不会探讨如何创建可视化内容/面板并保存它们。相反，我们将看到如何生成并查看报告。先保存面板，然后单击**报告**，单击**生成可打印的 PDF**，会显示如下消息：

Reporting：Dashboard generation has been queued. You can track its progress under Management.（Reporting：要生成的面板已在队列中，你可以在 Management 界面中跟踪进度。）

要下载/查看报告，请按照下列步骤操作：

（1）单击 **Management** 页面。

（2）选择 Kibana 页面中的 **Reporting**。

（3）将在 **Generated Reports** 中看到文档的名称，如果已添加到队列，可以看到其状态和下载按钮。

（4）查看下载的报告如下图所示。

> 只能为已保存的对象生成报告。

可以通过以下方式使用 Kibana 开放的 API 生成报告：

- 已保存的搜索结果的 Report：
  /api/reporting/generate/search/<savedsearch-id>
- 已保存的可视化的 Report：
  /api/reporting/generate/visualization/<savedvisualization-id>
- 已保存的面板的 Report：
  /api/reporting/generate/dashboard/<saveddashboard-id>

> 要使用 API 生成报告，需要发送 POST 请求而不是 GET 请求。

也可以使用_g 参数指定时间段。

> 如果不确定设置过的时间段，那么只需从已保存的搜索结果、可视化内容和面板 URL 中复制_g 参数。

可以让程序使用 API 自动生成 Dashboard 报告，使用以下 curl 命令：

```
curl -XPOST http://localhost:5601/api/reporting/generate/dashboard/
SampleDashboard --user elastic:changeme -H "kbn-version:5.1.1"
```

其中，SampleDashboard 被保存在面板的 ID 中，出于安全设置，user 被用来指定用户名和密码，-H 参数需要发送 Kibana 的版本，在我们的例子中是当前的版本号 5.1.1。

要下载报告，仍需遵循上述步骤。



ProductVsPrice

Count

900
800
700
600
500
400
300
200
100
0

Product1 Product2 Product3 Product3

Product.keyword: Descending

0
1,000
2,000
3,000
4,000
5,000
6,000
7,000
8,000
9,000
10,000
11,000
12,000
13,000

CountryvsPaymentType

Count

200
150
100
50
0

United States United Kingdom Canada Ireland Australia

Country.keyword: Descending

Visa
Mastercard
Amex
Diners

elastic

Page 1 of 1

> 要生成报告，有些用户可能需要在 Ubuntu 系统中安装其他依赖项。如果报告生成失败，并显示“Phantom immediately exited with：127”，则需要执行以下命令安装 libfontconfig 和 libfreetype6。
>
> ```
> sudo apt-get install libfontconfig
> sudo apt-get install libfreetype6
> ```

10.4 本章小结

在组织中设置 Elastic Stack 组件时，X-Pack 插件是必备工具。利用它可以掌握所有基本和关键的信息，无论是用户管理、角色和权限、事件的警报、生成报告以及使用 Graph 分析数据，还是监视节点和索引。第 9、10 两章总结了新版 X-Pack 插件及其组件的功能。

在下一章中，将了解最佳的实践范例和标准，并在工程生产环境中实际应用。

Chapter 11

第11章 最佳实践范例

在前一章中，讨论了 Elastic Stack 的一部分——X-Pack 插件提供的各种组件，详细探讨了每个组件，涵盖了组件提供的内容以及各种功能。

如果一直在跟进这些章节，就可能有很多问题。例如，如何开始或继续创建可扩展的系统，或者应该遵循哪些最佳范例来创建一个高效的系统。

本章将了解 Elastic Stack 中的一些最佳实践范例，以便在学习他人的经验后，能够用于生产环境。

在本章中，将介绍以下主题：

- 为什么需要最佳实践？
- 了解你的使用案例；
- 选择正确的硬件；
- 搜索和索引性能；
- 调整 Elasticsearch 集群；
- Logstash 配置文件；
- 重新索引数据。

11.1 为什么需要最佳实践范例

开始学习一个工具时，我们总是尝试在独立的机器上使用，从而获得专业知识，并可以对各种各样的事情进行实验。当我们将小范围的配置放在多个大型系统的配置上时，可能有很多事情会阻碍系统提高性能，或者无法提供在一个小型配置中获得的最佳结果。为了取得最佳效果，我们需要理解和遵循其他规范，从而获得更高的性能和更少的麻烦。在编程语言方面，我们具有最佳编码实践的概念，它描述了如何编写易于理解、维护和阅读的代码，如果其他人开始使用相同或相似的代码，则可以减少工作量。但是，在工具或组件中，却没有最佳实践中的死板规定。最佳实践取决于各种因素，如架构设计、使用的组件、底层硬件等，还取决于用户想要解决的使用案例。

为了创建可以轻松扩展和维护的高度可扩展的分析系统,并采用最佳实践策略,使我们可以处理面临的常见瓶颈,最佳实践就变得十分必要了。这里没有让你盲目学习的具体内容;然而,从最初基于许多人的经验开始,它提供了一个很好的参考。它可以帮助你了解应该关注的领域,或在平台中应该微调的参数。

在 Elastic Stack 中,有四个基本组件:

- **Beats**:数据传输工具;
- **Logstash**:处理数据;
- **Elasticsearch**:存储数据;
- **Kibana**:数据可视化界面。

在这四个组件中,Elasticsearch 需要深入了解,才能有最佳实践,因为它是用于存储数据的主要组件,并作为可视化数据的数据源。如果 Elasticsearch 的性能受限,它将影响整体的设置。作为基于 Web 的用户界面,Kibana 运行所需的资源最少。Logstash 用于获取和灵活处理大量的数据。Beats 用于获取数据并将其发送到 Elasticsearch 或 Logstash 的轻量级数据传输工具中。

11.2 了解你的用例

这是所有人在考虑最佳实践或使用 Elastic Stack 之前应该了解的基本信息。如果在了解 Elastic Stack 而不考虑用例的前提下就开始连接或处理随机数据,那么将无法根据需要导出正确的信息,将会一直遗留一些问题。例如,为什么选择这样的组件不使用其他软件或工具?因此,事先了解用例是非常重要的。

用例主要涉及要使用的组件、流入日志数据的关键性、高可用性集群需求,以及集中式日志记录系统需求等许多问题。如果用例包括分析应用程序的日志,则可以做出相应的决定来确定是否需要单个 Elasticsearch 集群,或者是否需要创建集中式日志记录系统。

开始了解用例时,出现的最大问题就是系统架构应该是什么?应该如何设计架构?或者什么工具/组件应该是它的一部分?我们经常花费大量时间弄清楚是否应该使用 Logstash 或 Beats。一个常见问题是使用 Beats 会增加另一层复杂度,并且需要额外的知识来开发、调试,而使用 Logstash 需要大量的内存,因为它使用的 JVM 是资源密集型的。要了解系统如何设计,这是需要权衡的,特别是在已经使用输出日志的系统的情况下;需要使用 Elastic Stack,同时不破坏或影响系统性能。

如果从许多关键的服务器拉取日志,当它们在服务器上托管应用程序时,最好使用 Beats 作为消耗内存更少、消耗资源更低的工具。但是,如果现有的服务器还有大量内存可用,那么就可以直接在系统上安装 Logstash,并将处理后的数据推送到 Elasticsearch,从而

降低了附加数据传输层的复杂度。

如果使用 Elastic Stack 处理其他人使用的数据，则无法承担丢失数据带来的损失。在某些情况下，使用 Beats 将数据推送到 Logstash 后，它最终会把数据存储在 Elasticsearch 中。如果正在推送和正在处理的数据有延迟的情况，那么可能有些数据会在不知情的情况下丢失。在这种情况下，需要使用消息队列工具，它可以在短时间内存储数据并且防止数据丢失。因此，可以根据需要使用各种消息队列，如 Redis/Kafka。

理解用例之后，接下来的最佳实践活动就是安装 Elastic Stack 的组件。使用默认设置和配置来安装组件时，最好重新访问配置文件，更改各种参数。

11.3 管理配置文件

从生产部署的角度来看，对配置文件的更改，是最基本也是最重要的。下面看看各种组件的配置文件。

11.3.1 Elasticsearch——elasticsearch.yml

默认情况下，Elasticsearch 中可以设置重要属性的值，例如集群和节点相关的集群名称、节点名称等。虽然没有必要设置，但自定义它们的名称还是很不错的。例如，应该指定节点名称，以便记住并跟踪节点统计信息。其中几个属性的说明如下：

- 修改以下属性，更改集群的名称：

```
#cluster.name: my-application
cluster.name: production-elasticstack
```

- 修改以下属性，更改节点的名称以便轻松识别新加入群集中的节点：

```
#node.name: node-1
node.name: elasticstack1
```

- 修改以下属性，更改 Elasticsearch 存储数据目录的位置：

```
#path.data: /path/to/data
path.data: /usr/share/elasticsearch/data
```

> 在 Windows 系统中，路径应使用正斜杠[①]，如：
>
> ```
> path.data:
> E:\installations\stack\elasticsearch-2.3.0\config
> ```

① 译者注：此处原文中为反斜杠，笔者认为这里存在问题，应为正斜杠。

- 修改以下属性，更改 Elasticsearch 日志存储目录的位置：

```
#path.logs: /path/to/logs
```

11.3.2 Kibana——kibana.yml

类似于 Elasticsearch 的 elasticsearch.yml，Kibana 的 kibana.yml 也应更新。使用 Kibana 时，重要的几点包括：

- 修改以下属性指定 Elasticsearch 所在机器的 IP，更改 Elasticsearch 实例的 URL：

```
#elasticsearch.url: http://localhost:9200
elasticsearch.url: http://<ip-of-machine>:9200
```

- 如果希望在 Kibana 打开时将用户重定向到任何其他页面，例如已保存的面板，可以修改以下属性：

```
#kibana.defaultAppId: "discover"
kibana.defaultAppId: "dashboard/<Name-of-Dashboard>"
```

打开 Kibana UI 时，指定保存的面板名称即可直接打开已保存的面板。

- 如果使用 SSL 保护从 Kibana 服务器到 Web 浏览器的请求，请更改密钥和证书的位置，修改以下属性：

```
#server.ssl.cert: /path/to/your/server.crt
#server.ssl.key: /path/to/your/server.key
server.ssl.cert: /usr/share/kibana/ssl-server.crt
server.ssl.key: /usr/share/kibana/ssl-server.key
```

类似地，可以设置其他 SSL 属性来验证 Elasticsearch。例如使用相同的 SSL 文件集（elasticsearch.ssl.cert 和 elasticsearch.ssl.key）；提供由 Elasticsearch 实例（elasticsearch.ssl.ca）的证书来源给出的证书，并指定是否验证 SSL 证书（elasticsearch.ssl.verify）。

- 修改以下属性，更改创建进程 ID 文件的目录位置：

```
#pid.file: /var/run/kibana.pid
pid.file: /usr/share/kibana/kibana-processfile.pid
```

11.4 选择正确的硬件

安装 Elastic Stack 时需要了解一些重要的指标，例如需要多少内存、需要多少磁盘空间、保持适当的 I/O 请求、资源消耗所需的 CPU 或内核、系统中的网络。安装 Elastic Stack 时，通常会对每个组件使用的资源量产生混淆。下面利用各种组件解决资源需求的问题。

11.4.1 内存

内存容量是影响应用程序性能的最重要参数之一。如果提供的内存容量低于预期，则应用程序可能会停止，失败或显示内存不足的错误。

内存大小很重要，因为内存被 OS 用于执行各种任务。因此，确定向 Elastic Stack 组件提供多少内存是至关重要的，这样，应用程序和操作系统性能才不会受到影响。

在 Elastic Stack 的上下文中，内存对于决定是否应该配置 Elasticsearch 和 Logstash 是至关重要的，因为它们需要进行大量的数据处理以及良好的处理单元和内存大小，这些都会消耗大量的资源。Elasticsearch 是作为 Elastic Stack 的中心点，因为数据从 Logstash 或 Beats 来，存储在 Elasticsearch 中，同时 Kibana 也可以使用 Elasticsearch 中的数据来执行可视化。

11.4.1.1 Java 堆大小

Java 程序在运行时会消耗一定容量的可用内存，而这项参数是配置 Elasticsearch 消耗内存大小的重要参数。当数据被推送到 Elasticsearch 时，它开始消耗更多的内存，因此设置最佳的 Java 堆大小变得很重要。如果 Java 堆已经被用完，那么日志将不会再被推送到 Elasticsearch，而 Kibana 将无法正确显示可视化/面板的内容。

Java 堆被用完时可以看到以下错误：

```
message [out of memory][OutOfMemoryError[Java heap space]]
```

最佳做法是增加 Java 堆的容量，默认设置为最大 1 GB。是的，应该增加它，但这个设置的最优值应该是多少，而不会对应用程序和系统的性能产生负面影响？我们来看一些最好的做法：

- 将堆的最大值和最小值设置为相等：
  在运行过程中不需要额外分配内存的情况下，这项设置是很有用的。
- 如果系统的总内存容量小于或等于 64 GB，则提供内存总量的 50% 为 Java 堆的容量：
  由于操作系统需要内存才能执行其他操作，因此建议最多为 Java 堆提供总内存容量的 50%。
- 如果系统的总内存大于 64 GB，则为 Java 堆提供 32 GB 的内存。即使拥有大内存的系统，也不要设置超过 32 GB 限制的内存容量。当 Java 堆使用 32 GB 或更多容量时，JVM 需要更多内存的 64 位指针来保存对象。当 Java 堆使用 31 GB 或更少时，JVM 使用 32 位指针，这种指针使用普通对象指针(Ordinary Object Pointers)压缩而成。

可以通过两种方式来设置 Java 堆的容量：

- 执行以下命令，在启动 Elasticsearch 节点时，同时提供 Java 堆的最小值和最大值：

```
./bin/elasticsearch -Xms8g -Xmx8g
```

其中，-Xms8g 将最小值设置为 8 GB，-Xmx8g 将最大值设置为 8GB。

> 注意：如果使用的是 32 位系统，则堆的最大值是很小的。在 Windows 系统中，堆的最大值约为 1.5 GB。在 UNIX 系统中，堆的最大值为 2 GB。在运行 32 位虚拟机的 64 位操作系统中，堆的最大值可以设置为 4 GB。

更改位于 $ES_HOME/bin(UNIX 系统)或 $ES_HOME\bin(Windows 系统)中的 elasticsearch shell 脚本文件(elasticsearch.in.sh)中的以下配置：

```
ES_MIN_MEM=2g
ES_MAX_MEM=2g
```

> 需要重新启动 Elasticsearch，才能应用这些修改过的设置。

11.4.1.2 交换空间

交换空间是将内存页从内存中复制到硬盘中，以释放内存的过程。它将页面从高访问速度的内存转移到低访问速度的磁盘。交换空间启用时，Elasticsearch 的运行性能会大打折扣，这对 Elasticsearch 来说是个不利因素。交换次数越多，程序的执行速度就越慢。

Elasticsearch 已经在 elasticsearch.yml 配置文件中提出了以下建议：

```
Elasticsearch performs poorly when the system is swapping the memory.
```

(当系统正在启用交换空间时，Elasticsearch 会表现不佳。)

可以在 elasticsearch.yml 文件中设置以下属性来限制内存的交换：

```
bootstrap.memory_lock:true
```

> 需要重新启动 Elasticsearch，才能应用这些修改过的设置。

可以执行以下命令，验证设置是否已应用：

```
curl http://localhost:9200/_nodes/process?pretty
```

或者，可以在浏览器中输入以下 URL 来执行相同的命令：

```
http://localhost:9200/_nodes/process?pretty{在浏览器中}
```

其中，localhost 是 Elasticsearch 节点的主机名。

> 如果将其设置为 true 后 memory_lock 的值是 false，则表示请求失败。第一个原因可能是用户没有锁定内存的权限，可以通过运行 ulimit -l 来解决。也就是说，在启动 Elasticsearch 之前，以 root 身份来解除限制。第二个原因可能是 temp 目录有 noexec 设置。这可以通过在 Elasticsearch 开始运行时指定一个新的 temp（临时）目录来解决，方法为. /bin/elasticsearch-Djna. tmpdir = /new/path/to/new/dir。

以上设置是使用 Elasticsearch 和 Logstash 的最佳做法。

对于 Kibana 和 Beats 来说，其内存需求是最小的，因为它们运行时消耗内存非常少。Kibana 作为基于 Web 的客户端，它根据浏览器的消耗来确定需要多少内存。占用内存空间很低的 Beats 组件占用最少的内存来获取数据，并发送到 Logstash 或 Elasticsearch。

11.4.2 磁盘

由于 Elasticsearch 是数据存储的唯一组件，因此搭载数据的磁盘也成为系统中一个重要的部分。如果磁盘空间已满，则其中将不会存储新数据，节点可能会停止、运行失败或关闭。当处理副本时，如果节点出现故障，则其他节点必须具有容纳其副本和数据的能力，才能很好地规划存储。此外，对于需要高速索引的集群或节点，以及获取大量日志数据的能力而言，磁盘都显得非常重要。在这种情况下，当涉及基于大量写入操作时，磁盘可能会发生故障，并成为集群的瓶颈。

一些最佳做法是：

- 如果读/写索引和数据的操作有所增加，则可以使用 SSD 硬盘。
- 使用高 RPM（15 K RPM 驱动器）的高性能磁盘。
- 通过有效利用 RAID 0 的实现来提高磁盘速度。
- 使用本地存储来存储数据，而不是 NAS 或其他实现形式。

Elasticsearch 需要磁盘空间来存储数据、分片和副本。规划磁盘大小的做法没有固定的规则。可以执行以下命令，从以下 Elasticsearch 属性中获取一些提示。

- cluster. routing. allocation. disk. watermark. low：指定磁盘可以继续被占用的最大容量限度。默认情况下，它的值设置为 85%。也就是说，在一个节点的磁盘使用量超过 85%时，这一节点上不会再分配分片。该值可以设置为百分比或绝对字节值（如 500 MB 或 10 GB）的形式。
- cluster. routing. allocation. disk. watermark. high：指定磁盘可以继续保存数据的最大容量限度。默认情况下，它的值设置为 90%。也就是说，在一个节点的磁盘使用

量超过 90％时，这一节点上的分片将会被重新分配到其他节点。该值可以设置为百分比或绝对字节值(如 500 MB 或 10 GB)的形式。

这里提到的百分比值是指已使用的磁盘空间，而字节值是指空闲的磁盘空间。

- cluster.routing.allocation.disk.threshold_enabled：指定是否使用磁盘容量阈值。如果设置为 true，则上述限度将适用，否则将不适用。默认情况下，其值被设置为 true。这项属性也可以使用集群 API 进行设置。

以下是使用基于 REST 的 API 调用 Elasticsearch 的示例：

```
PUT /_cluster/settings
{
    "transient" : {
        "cluster.routing.allocation.disk.threshold_enabled" : true
    }
}
```

在位于$ES_HOME/config(UNIX 系统)或$ES_HOME\config(Windows 系统)的 elasticsearch 配置文件(elasticsearch.yml)中添加以下属性：

```
cluster.routing.allocation.disk.threshold_enabled: true
cluster.routing.allocation.disk.watermark.low: 80%
cluster.routing.allocation.disk.watermark.high: 85%
```

或者：

```
cluster.routing.allocation.disk.watermark.low: 15gb
cluster.routing.allocation.disk.watermark.high: 10gb
```

> 重新启动 Elasticsearch 后才能应用这些修改过的设置。

对于磁盘的适当大小，使用了默认值，并通过某些测试提出了一个公式，如下所示：

$$(\text{space per node} \times 0.85) \times ((\text{node count} - \text{replica count}) / \text{node count})$$

假设节点包含 1TB 的磁盘空间，ES 集群有 5 个节点，副本数量设置为 2。为了避免因 5 个节点中有 2 个节点掉电而引发问题，每个节点都应该允许存储一定容量的数据，其计算公式如下所示：

$$(1\text{TB} \times 0.85) \times ((5-2)/5) = 510\ \text{GB}$$

> 上述计算细节可以从以下资料中获得：
> http://svops.com/blog/elasticsearch-disk-space-calculations/

11.4.3 输入输出

输入输出(I/O)调度是最不被关注的参数,但默认情况下,它提供了良好的性能。在使用 UNIX 内核的情况下,如果使用 SSD 硬盘而不是 HDD,则需要对该参数进行调整。

默认情况下,I/O 调度是**完全公平队列(Completely Fair Queuing,CFQ)**。这种类型的调度机制在处理提交信息时提供请求,并随后为每个队列分配时间片,以同步的方式访问磁盘。这种调度机制非常适合 HDD 硬盘的运转方式。

对于 SSD 硬盘而言,由于没有盘片旋转的运转机制,这种调度机制效率很低。对于 SSD 硬盘,可以使用**空操作(Noop)**或**截止时间(Deadline)**的方式:

- Noop 是最简单的 I/O 调度器,它将请求放入基本 **FIFO(先进先出)**队列中。
- 截止时间的方式保证每个请求的服务启动时间。此调度程序可以降低 I/O 延迟,提高吞吐量,并消除重新将 I/O 请求排序所需的时间。

可以在 UNIX 环境中执行以下命令更改设置:

- 检查当前执行的调度程序:

  ```
  cat /sys/block/sda/queue/scheduler
  ```

  返回的结果中,将在方括号中显示各种调度程序和选定的调度程序名称,如下面的命令所示:

  ```
  noop anticipatory deadline [cfq]
  ```

- 更改调度程序:

  ```
  echo noop | sudo tee /sys/block/sda/queue/scheduler
  ```

- 检查调度程序是否已更改:

  ```
  cat /sys/block/sda/queue/scheduler
  ```

  返回的结果中,将在方括号中显示各种调度程序和选定的调度程序名称,如下面的命令所示:

  ```
  [noop] anticipatory deadline cfq
  ```

> 这是一个暂时的更改,并将在机器重新启动后重置调度程序。此外,Noop 不适用于 HDD 硬盘。

11.4.4 CPU

Elastic Stack 组件往往对所需的 CPU 资源要求较低。与其他资源相比,其要求较少。最佳做法是:

- 选择现代处理器;
- 选择多个核心;
- 如果有选择 CPU 与内核方面的困难,则选择具有较高内核数量的系统,而不是具有较高时钟速度的 CPU。

多核 CPU 具有许多优点,因为任何操作(如索引或搜索)都使用一个 CPU 内核,所以多个内核将有助于执行多个并发操作。

11.4.5 网络

有些人可能忽略这个问题了,因为每个人都能提供或使用快速且优质的网络连接和设备。要在分布式系统中实现良好的性能,需要一个快速可靠的网络。高带宽对各方面都有好处,并且低延迟确保了节点可以无缝通信。

一些最佳做法是:

- 使用现代数据中心,完成网络连接;
- 使用高速连接,完成节点连接;
- 避免在跨越地理位置上距离太远的数据中心构建集群;
- 尝试在一个数据中心内集成一个集群;
- 如果使用多个数据中心,确保多数据中心之间的网络连接稳健,并保证低延迟。

11.5 搜索和索引性能

到目前为止,已经发现了一些资源利用方面的最佳做法。内存、CPU、I/O、磁盘和网络在选择首选系统配置的过程中起着重要的作用;可以调整一些设置,以改善在 Elasticsearch 和 Lucene 中执行搜索和索引时的资源使用情况。

11.5.1 过滤缓存

默认情况下,Elasticsearch 中用于查询的过滤器被缓存时,意味着查询使用过滤器时 Elasticsearch 找到与过滤器相关的文档,并将过滤器存入缓存中。缓存后,每当具有相同过滤器的查询被执行时,它将提供更快的结果,因为过滤器内容已被缓存到内存。由于内部使用内存,因此设置属性,以限制 Filter 高速缓存的使用,这是十分明智的。虽然每个过滤器

使用较少的内存,但如果使用大量的过滤器,则 JVM 堆的大小可能会受到影响。使用以下属性,可以限制过滤器高速缓存使用的堆内存容量:

```
indices.cache.filter.size:10%
```

默认情况下,这项属性的值为 10%。它可以接受百分比或内存字节级别的值,例如 256MB。如果要使用大量的过滤器,那么可以将此属性的值增加到适当的值。在 elasticsearch.yml 配置文件中可以添加此属性。

11.5.2 Fielddata 的容量

这是一个重要的设置,它直接影响 Elasticsearch 的搜索和执行查询的性能。一旦缓存了查询、搜索或字段,就能够通过直接加载缓存内容来提供更快的结果。缓存所有内容可能会产生一个问题,它取决于可用的缓存容量,一旦系统中的缓存已满,则缓存将会导致搜索和索引数据的性能不佳。

Fielddata 可将数据从磁盘换入内存,这是十分消耗系统资源的。如果 fielddata 超过指定的大小,则缓存中的数据需要被换出,以便为新数据创建空间,这是一项非常消耗系统资源的操作。如果 Elasticsearch 需要反复将数据从缓存换出,并将数据重新加载到缓存中,那么性能将会很差。

以下属性可用于限制 fielddata 的大小:

```
indices.fielddata.cache.size
```

默认情况下,这项属性的值是无界的,这意味着 Elasticsearch 将永远不会从 fielddata 中删除数据。想象一下日常创建索引的场景,通常会保留所有的索引。Fielddata 将不断增长,默认情况下不会从缓存中取出数据。在这种情况下不会加载新的数据,此时 fielddata 将被完全充满,程序将被卡住,因为无法加载新的数据。如果不常使用旧索引,建议将属性设置为清除旧数据。

可以在 elasticsearch.yml 配置文件中添加此属性:

```
indices.fielddata.cache.size: 40%
```

这项属性用来设置 fielddata 缓存容量的最大值,以百分比或字节值来定义节点堆空间(node heap space)的大小。

除了该属性,还需要了解断路器的工作。为什么要了解这个属性?这是一个有趣的问题。一旦数据被加载,Fielddata 总是会被检查。如果其大小可满足需求,就会将数据加载到缓存中,否则会清除旧数据并重新加载。如果一个查询尝试将更多的数据加载到 fielddata 时超过了可用内存会怎么样?在这种情况下,将得到 OutOfMemoryException 错

误提示。

为了处理这种情况，断路器基于各种参数估计一个查询所使用的内存，并检查内存容量是否可用。如果估计查询的要求高于阈值，则断路器将被触发，查询将异常中止。这样，OutOfMemoryException 错误提示便不会出现。

可以在 elasticsearch.yml 配置文件中添加此属性：

```
indices.breaker.fielddata.limit: 70%
```

它将按照节点堆空间的百分比或字节值来设置 fielddata 缓存容量的最大值。默认情况下，此属性的值为 60%。

> ℹ请确保 fielddata 缓存容量的值低于 fielddata 断路器的阈值。如果缓存容量高过阈值，断路器将被触发，将会得到以下异常信息：
>
> ```
> ElasticsearchException [ org. elasticsearch. common. breaker.
> CircuitBreakingException: Data too large, data for field [id] would be larger
> than limit of [8589934592/8gb]];
> ```

上述设置也可以使用 REST API 命令动态设置：

```
PUT /_cluster/settings
{
    "persistent" : {
    "indices.breaker.fielddata.limit" : "50%"
    "indices.fielddata.cache.size" : "40%"
    }
}
```

11.5.3 索引缓冲区

索引缓冲区用于存储新索引的文档。当缓冲区已存满时，索引中的文档将会写在磁盘上，由于与内存相比，读取磁盘数据的速度会很慢，从而会导致性能低下。因此，增加索引缓冲区的大小很重要。需要进行以下设置：

```
indices.memory.index_buffer_size:30%
```

默认情况下，设置的值为 10%，这意味着索引缓冲区占堆内存总量的 10%，该值可以设置为百分比或字节值。该属性需要添加在 Elasticsearch 的配置文件 elasticsearch.yml 中。

> ℹ这是一个静态属性，必须在集群中的每个数据节点上进行配置。

11.6 调整 Elasticsearch 集群

选择正确的节点类别,对于确定集群中的节点数量、要使用的分片和副本的数量、要存储的索引数量,从而有效地规划 Elasticsearch 集群是非常重要的。确定 Elasticsearch 集群的大小没有固定的规则。

11.6.1 选择正确的节点

在 Elasticsearch 中处理节点时,对于可用的不同类型的节点没有明确区分。让我们来了解 Elasticsearch 集群中不同类型的节点。

11.6.1.1 主节点和数据节点

主节点和数据节点(master and data node)是 Elasticsearch 实例启动时,在 Elasticsearch 集群中创建的默认节点。这种类型的节点充当主节点并存储数据。如果此节点不是主节点,那么当另一个节点发生故障时,此节点将可以成为主节点。它执行主节点和数据节点的操作,如集群相关任务,创建或删除索引,跟踪集群中的节点等。该节点消耗大量资源,因为它需要主节点操作和数据节点操作的资源。因此,在类似的生产环境中,如果在该节点上的负载很高,那么该节点可能会停止。

要配置主节点和数据节点,请在 elasticsearch.yml 中添加以下属性:

```
node.master: true
node.data: true
node.ingest: false
```

1. 主节点

这是在 Elasticsearch 中的一种类型的节点,它仅执行主节点操作。这种类型的节点在生产环境中非常必要,因为它不会消耗大量资源,并且不执行资源密集型任务。此外,为了避免先前解释的问题,要明确区分主节点和数据节点并保持集群的稳定性,这一点很重要。在生产环境中,让主节点处于始终稳定的状态是非常重要的。

要配置主节点,请在 elasticsearch.yml 中添加以下属性:

```
node.master: true
node.data: false
node.ingest: false
```

> ℹ主节点(master nodes)和主数据节点(master data nodes)也称为 master-eligible node。

2. 数据节点

这是 Elasticsearch 中另一种仅在节点内存储数据的节点。除了存储、索引和提供搜索查询结果之外，它不执行其他操作。它可以被认为是一个工作节点，该节点执行与 CRUD（增、删、改、查）、聚合和搜索之类的各种数据相关的操作。该节点完全不同于主节点。

要配置数据节点，请在 elasticsearch.yml 中添加以下属性：

```
node.data: true
node.master: false
node.ingest: false
```

3. Ingest 节点

这是 Elasticsearch 中近期推出的另一种类型的节点，用于在存储数据前对事件进行预处理。它由单个或多个 processor 组成，可以执行处理事件的预处理流水线。

要配置 Ingest 节点，请在 elasticsearch.yml 中添加以下属性：

```
node.data: false
node.master: false
node.ingest: true
```

4. 非主节点、非数据节点以及 Ingest 的其他节点

这是 Elasticsearch 中另一种类型的节点，它们既不用作主节点、数据节点，也不用作 Ingest 节点。你可能想知道一个节点不执行 Elasticsearch 节点基本操作的目的是什么。该节点充当了负载均衡器，适当地为请求分配路由。它将所有的集群级请求的路由分配到主节点以及与数据节点所有数据相关的请求中，并将负载分配到集群中的其他节点中。它作为主节点和数据节点之间的智能协调器，是仅用于协调的节点。

要配置这个仅用于协调的节点，请在 elasticsearch.yml 中添加以下属性：

```
node.master: false
node.data: false
node.ingest: false
```

> ℹ 在集群中添加大量仅用于协调的节点会增加系统负担，因为主节点跟踪的每个节点都是集群的一部分，而它们同时也需要获取自己这一部分中每个节点的确认信息。

集群中的每个节点除了各自的角色外，还可以处理 HTTP 请求和 Transport 的网络流通量。Transport 是一个能够无缝连接节点之间内部通信的模块。使用 HTTP 请求，可以

使外部 REST 客户端访问 HTTP 层或通过 HTTP 层访问 API。

要启用 HTTP 请求，请在 elasticsearch.yml 中添加以下属性：

```
http.enabled: true
```

要禁用 HTTP 请求，请在 elasticsearch.yml 中添加以下属性：

```
http.enabled: false
```

在数据节点上禁用 HTTP 请求，可以确保数据节点仅执行与数据相关的操作，而不会因此而造成负担。

> 应该在与 Kibana 的主机相同的机器上运行 Client Node，它可以轻松地在作为负载均衡器的节点上分发 Kibana 请求。

11.6.2 确定节点数

每个人都想知道如何估计 Elasticsearch 集群的节点数。其实这很难估计，因为它涉及各种因素，例如：

- 每个节点有多少内存？
- 每个节点有多少磁盘空间？
- 将创建多少索引？
- 将在一天或更短的时间内存储多少数据？
- 想创建多少个分片？ 每个分片要创建多少副本？
- 会使用多少搜索查询？

在估计集群中的节点数量之前，所有上述问题似乎都被忽略，或者无法清楚地回答。如果不知道或无法确定所需的节点数量，则从少量节点开始。

假设正在使用 Elastic Stack 进行日志分析，其中包含每天消耗 500 MB 数据的一百万条记录。创建一个按天存储的索引，并且从 Kibana 面板执行少于 100 个搜索查询。此外，不使用任何副本。在这种情况下，应用程序不是很重要。可以选择单一的节点，内存为 64 GB，磁盘空间为 1 TB，并具有四个 2.2 GHz 的 CPU 核心。对于这种情况，一个节点就足够了，直到磁盘空间存满为止。

避免使用两个节点，这样会出现脑裂(Split-Brain)的问题，这个问题在配置不当的集群中非常常见。这种情况是由于某些原因导致节点之间的通信不正确，或节点发生故障而引发的。

如果其中一个从节点不能与主节点通信，或者该节点是主节点，则会选出新的主节点。新的主节点将执行故障主节点的任务。一旦故障主节点恢复，新的主节点就将再次成为从

节点。在 Split-Brain 问题中，假设有两个节点，如果主节点关闭，则从节点将成为主节点。一旦故障主节点恢复，则出现以下情况：

- 故障的主节点认为新的主节点将被降级为从节点。
- 新的主节点认为原始主节点掉线，并应作为从属节点再次加入。

因此 Split-Brain 问题发生了。

防止它发生的最佳做法是：

- 在 Elasticsearch 集群中运行奇数个节点。
- 将每个节点分别手动配置为主节点或数据节点，以避免在 Elasticsearch 集群的小范围设置中出现 Split-Brain 问题。
- 配置 discovery.zen.minimum_master_nodes 属性。这项属性可以在选择新的主节点之前提供集群中需要运行的最少节点数。
- 配置 discovery.zen.minimum_master_nodes 的公式是 $(N/2)+1$，其中 N 是集群中的节点数。

11.6.3 确定分片数

这是一个非常困难的问题，因为分片的数量取决于各种因素，如系统的内存、磁盘空间、索引数量、索引中的文档总数、文档占用的大小、运行查询的数量、搜索复杂度、数据的聚合等。

面对如此大量的依赖因素，无法评估一个分片是否太少或数千个分片是否太多。另外，分片一经定义就不能更改，除非重新索引文档，而这会带来额外的系统开销。

要确定分片数量，一个关键参数是了解数据量将会增长多少。每个分配分片的方案都会有附加成本，因为分片消耗 CPU、内存、硬盘和文件处理功能等各种资源。每个调整集群大小的决定都会落到这个问题上：试图解决的用例是什么？因为一切都取决于用例的重要性和关键性，这将使得它更容易规划。

以下是一些最佳做法：

- 将分片数设置为集群中节点数量的一到三倍；
- 不要设置大量分片来过度使用集群；
- 通过在测试环境中创建类似生产的场景，填充实际数据并执行拟使用的搜索查询，来了解分片的最大处理能力；
- 如果集群的性能由于添加了分片而降低，则添加的节点和分片将由 Elasticsearch 集群自动均衡；
- 从较少的副本开始，然后随着时间的推移而增加。

如本章前面所讨论的那样，使用分片和副本计算磁盘空间时，重要的是相应地调整分片

的大小，因为一切都将取决于此。在定义了可以由单个分片处理的容量之后，可以通过设置额外的缓冲区，并观察数据的瞬时增长，来推算分片大小。如果数据增加，可以用总数据量和附加缓冲区的大小来除以单个分片的总容量，以此作为设置分片大小的依据。

11.6.4 缩减磁盘空间

磁盘空间是确定节点或集群大小的重要参数之一，可以减少索引和文档所占用的磁盘空间，这点很重要。这可以通过配置级属性完成，并在索引级别进行一些更改。我们来看看一些最好的做法：

- 通过将以下属性添加到 elasticsearch.yml 文件来启用压缩：

```
index.store.compress.stored: true
index.store.compress.tv: true
```

上述属性用于压缩索引中的存储字段，并压缩词项向量文件，该文件存储了文档字段中有关词项的统计信息。

- 删除所有不必要的字段或仅为必需的字段创建索引。此步骤可以在 Logstash 级别完成。
- 如果不需要，请删除@message、@source 和其他字段。如果使用 Logstash Grok 过滤器，则可以在各个字段中分解消息字段，因此可以删除@message 字段。
- 如果不需要，请禁用_all 字段。

  这个字段是将所有字段的值连接成单个字符串的特殊字段，它需要额外的磁盘空间和 CPU 资源才能运行。

  这里存在一个需要权衡的问题。启用该字段后，搜索一个词时，即使不了解该词在哪个字段中存在，也可以得到搜索结果；禁用该字段后，就需要知道确切的字段名称并指定要搜索的内容。
- 将字段设置为 not_analyzed，再根据需要添加 analyzed 字段。

  这是确定搜索如何工作的另一种特殊字段属性。它消耗额外的资源来分词，如内存和磁盘空间。默认情况下，如果词包含点、连字符等内容，则将其分割成多个子词。

  这里也存在一个需要权衡的问题。有些词项需要精确完整地表达而不可拆分，在这样的使用场景下，将字段设置为 not_analyzed 总是有好处的。

11.7 Logstash 配置文件

配置文件是运行 Logstash 的关键。由于 Logstash 需要创建/更新配置文件，所以重要的是要有一个有效和灵活的配置，可以在需要时轻松地进行更改。我们来看看应该遵循的

一些原则。

11.7.1 对多个数据源分类

从多个不同的数据来源收集数据(以便为随后的数据分析做准备)时,最好为每种不同来源的数据分别创建一个类型,对其进行分类。

我们来看下面的例子:

```
input {
    file {
        path =>"/path/to/directory/"
        type =>"datanode"
    }
    file {
        path =>"/path/to/directory/"
        type =>"hbase"
    }
    file {
        path =>"/path/to/directory/"
        type =>"yarn"
    }
}
```

添加类型时,可以根据其关联的数据源类型使用不同的过滤器和输出。

11.7.2 使用 conditional 条件

Logstash 的配置应该基于不同的使用场景,在过滤日志信息时使用 conditional 条件。它提供执行某些既定操作的条件,而这些操作将匹配在条件上。有不同来源的数据需要进行特定过滤或输出时,应该使用这种条件。我们来看下面的例子:

```
output {
    if [type] =="datanode" {
        stdout { codec =>rubydebug }
    }
    if [type] =="hbase" {
        elasticsearch {
        hosts =>"localhost:9200"
        index =>"logs-%{+YYYY.MM.dd}"
        document_type =>"hbase"
        }
```

```
        }
        if [type] =="yarn" {
        file {
            path =>"/var/log/logstash/file.txt
            }
        }
}
```

11.7.3 使用自定义 grok 模式

grok 模式比以往任何时候都更强大。对于解析数据源，如果数据带有自定义格式或 grok 原本不存在的格式，那么建议创建自定义 grok 模式。拥有自己定制的 grok 模式，处理和解析数据时可以有更多的灵活性。

11.7.4 简化 grokparsefailure

每当 grok 模式无法匹配时，就会自动添加_grokparsefailure 的标签。但是，如果有多个 grok 模式，则很难知道哪个 grok 模式解析失败。为此，为每个匹配的 grok 模式添加 tag_on_failure 标签，以便容易识别是哪个 grok 过滤器解析数据失败。

例如：

```
grok {
    match =>["message","%{PATTERN_NAME}"]
    tag_on_failure =>["grokfailed_patternname"]
}
```

11.7.5 字段的映像

Logstash 解析并处理所有字段，然后将其作为字符串类型数据存储在 Elasticsearch 中。建立字段的映像有两种方法：

- 使用 Logstash 智能模式：用于提供 grok 模式中的字段显式映像。例如：

```
%{GREEDYDATA:fieldname:float}
```

可以指定 float 或 int 来建立字段的映像。

- 使用 Mutate 过滤器和 Convert 属性。

11.7.6 动态模板

它用于创建模板，以便根据数据和模板中定义的数据类型来解析所有字段，并执行操

作。默认情况下，使用 Logstash 将数据推送到 Elasticsearch，并且数据的索引与 logstash-* 匹配时，程序会自动使用模板。此模板找到一个字符串数据类型的字段，并创建两个字段：fieldname 和 fieldname. keyword，其中 fieldname 经过分词，而 fieldname. keyword 不进行分词。可以创建自己的动态模板，这样就不用再显式设置字段的映像了。

11.7.7　测试配置

建议测试 Logstash 配置，以检查配置文件是否正确，以及数据解析是否正确。下面是两个不错的方式：

- 使用--configtest 选项检查配置以及配置的语法是否正确。
- 使用 stdout 输出验证数据是否正确解析。

此外，还可以使用 stdin 输入和 stdout 输出验证用于处理数据的逻辑是否正确。

11.8　重新索引数据

更改字段的模式（schema）或映像时，Elasticsearch 中对数据的重新索引是一个挑战。更改模式时，需要重新索引该字段的所有文档，以便将映像的更改应用到先前存储的文档中；或者不重新索引旧文档，这些旧文档将随着模式的更改而失去作用。

重新索引数据的过程如下：

(1) 使用新的映像和设置来创建一个新索引。

(2) 从旧索引中取出文档，并存入新索引中。

应尽可能减少更改模式和重新建立索引带来的影响和停机时间，请使用下面的方法。

别名（alias）是一个强大的功能，利用它可以轻松地将完整的索引数据重新索引，而无须任何停机时间。别名相当于给出索引名称的昵称，它可以被认为是一个象征性的链接。

让我们看看如何使用别名：

- 使用文档的映像和设置创建索引；
- 创建一个别名来指向索引名称。

更新模式/映像之后：

- 使用新的映像和设置创建一个新索引；
- 使用重新索引 API，将旧索引的数据重新索引到新索引中；
- 删除旧索引别名，并将新索引添加到已创建的相同别名中；
- 删除旧索引。

11.9 本章小结

在实践中应该遵循最佳实践范例，因为这些范例可以消除设置、配置和使用 Elastic Stack 时面临的各种问题。可以根据要求调整多种设置，使 Stack 运行更加稳定。

在本章中，我们讨论了许多配置和使用 Elastic Stack 的方法。虽然本章尽可能涵盖大部分要点，但也可能还有其他一些设置是满足特定要求的最佳做法。必须仔细分析配置和设置，以避免出现漏洞。请记住，只要有一个糟糕的设置，就可能导致一场灾难。

下一章将介绍一个案例研究，探索如何利用 Elastic Stack 来实现最终目标。



案例分析——Meetup

前面已经介绍了 Intranet portal 的实际使用场景，其中展示了第 6 章 Elastic Stack 实战中 Elastic Stack 的强大功能。这个案例更多的是日志管理而不是数据分析。在本书中，我们已经了解到，Elastic Stack 在数据分析方面可以发挥出神奇的功力。在 Kibana 的帮助下，可以为数字赋以丰富多彩的表现形式，这些最终可以帮助你深刻理解数据，以此来高效配置整个应用系统环境，并准确做出决策。

在本章中我们来了解 Meetup，并分析一个真实的使用场景。我们将收集 Meetup 数据，找出各类统计数据、热门 Meetup 等信息。本章将介绍的主题如下：

- 了解 Meetup 的使用场景；
- 环境搭建；
- 使用 Kibana 分析数据。

12.1 了解 Meetup 使用场景

世界可能正在向核心家庭转移，但是人们每天的聚会(meetup)却越来越多，这表明人们仍然需要关注彼此的互动。Meetup 是一个用于人们聚会的应用。人们会时常举办活动、做事情、分享知识，并计划共同的兴趣。这是由 Meetup 公司于 2002 年 6 月推出的一个软件平台所支持的。该平台可以在 https://www.meetup.com/上访问，并可供大家加入，以及参加聚会。它在全球范围内有超过 2900 万用户、超过 180 个国家的 26 万多个团体，人们平均每月会参加超过 45 万次聚会。通过这些，完全可以感受到这款应用是多么受欢迎。

这些聚会并不仅限于谈论技术，人们还会为徒步旅行、登山、旅游、环境相关、冥想等活动聚集在一起。所有这一切为我们提供了一个绝妙的机会，分析世界各地举行的聚会。当然，类似的事情也发生在其他渠道组织的活动中。

那么问题是，我们要获取什么？怎样获取呢？我们所需要的就是从 meetup.com 收集数据的参数(endpoint)。具有丰富功能的 Meetup API 能使我们获取有关 Meetup 的数据，例如 Meetup 的群组、类别、活动以及为 Meetup 完成的 RSVP 等。目前有许多可用于不同

版本 API 的参数。如果有兴趣了解 Meetup API，可用的文档就在 https://www.meetup.com/meetup_api 中。只有很少一部分端点是公共的，可以不经授权就调用；其余部分需要提供 API 密钥。本章稍后将介绍如何调用这些端点。将获取的数据归为如下几类：

- Meetup 的类别；
- 热门城市；
- 群组；
- 开放的活动；
- RSVP 统计；
- 话题；
- 活动场所。

用于分析的数据不限于此。我们只选择了可以公开获得或适用于更大体量的数据，而没有获取任何用户或活动的具体内容。

12.2 环境搭建

就像任何其他情况一样，需要设置 Elasticsearch、Logstash 和 Beats(其一或两者)，以及用于可视化的 Kibana。需要一些特定的东西才能从 Meetup 那里获取数据，棘手的事情是用 Logstash 还是 Beats？或两者的组合？

有一个为 Logstash 开发的输入插件，可以读取 Meetup 数据，但这对我们来说还不够，因为我们想要读取更多的数据，因此需要另外选择工具。这里也没有合适的 Beat 可从 Meetup 获取数据。

现有的插件可以访问如下网站来获取：

```
https://github.com/logstash-plugins/logstash-input-meetup/
```

Elastic 官网也在如下网站：

```
https://www.elastic.co/guide/en/logstash/5.1/plugins-inputs-meetup.html
```

中记载了这个插件。这种插件使用以下参数之一来获取 Meetup 活动数据：

- venue_id：可以指定多个 ID，用逗号分隔。
- group_id：可以指定多个 ID，用逗号分隔。
- group_urlname：到 meetup.com 的群组的路径。

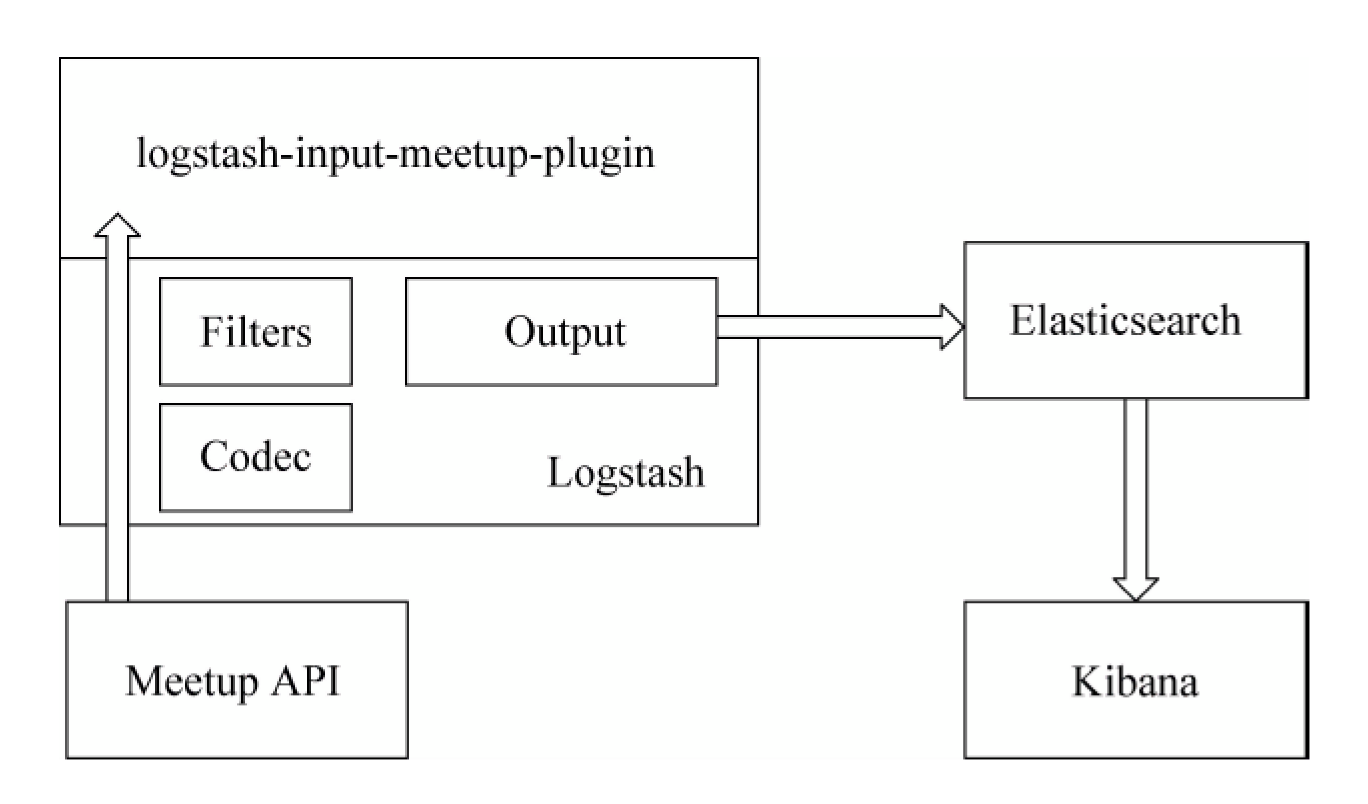
这个插件带来的限制是只能为所选的 place_ids、group_ids 等参数获取数据。选择它们是因为要知道从哪些群组或场所收集数据。但是，我们想做更多的事情(如上一节所述，我们希望为 Meetup 群组、活动、场所以及类别、主题和其他相关详细信息获取数据)。我们想

分析整个城市、整个国家的数据，或者可能是世界的数据，看看发展趋势如何。类似于Logstash中的twitter插件（它允许使用关键字进行搜索，然后收集推文）；但是如果想要获取所有推文的流式传输，则需要更改插件代码。当现有的代码不能用于满足新的要求时，已实现的插件需要更改才可以派上用场。

因此，我们不得不去寻求另一种解决方案——可以获取更多数据的解决方案，从而可以获取城市数据并生成大量事件来进行分析。

我们需要创建一个Beat或Logstash输入插件来满足需要。如前所述，Meetup API可用于收集数据。为了保持简单的架构，并在分析过程中关注更多信息，在本章中，将为Logstash编写一个输入插件，并跳过使用Beats的步骤。

最终，我们的架构如下图所示①。



如果不得不做以下事情，我们会转而开发Beats组件：

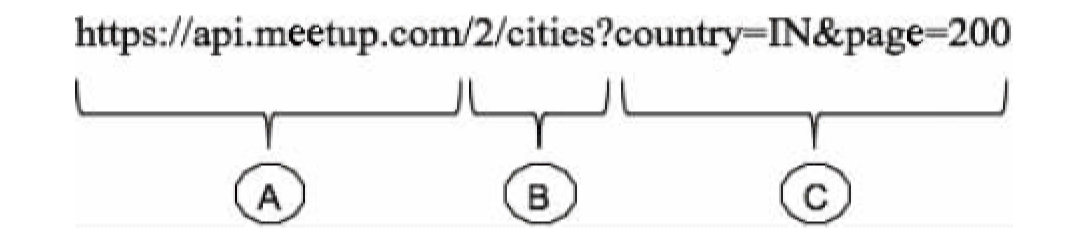
- 从多个服务器收集数据；在这种情况下，轻量级的Beats组件可以提供很多帮助，但是我们不会从多个服务器收集数据。只有调用Meetup API才能收集数据，因此不需要多个数据接收端。
- 做额外的处理，设计字段，接着使用Beats组件来收集数据，并使用Logstash进行处理。Logstash的插件能同时负责数据的收集和处理，因此不再需要Beats组件。

我们将开发用于Logstash的插件，它将使用Meetup API读取数据；我们将解析数据以准备事件，并将其发送到Elasticsearch索引中，然后使用Kibana面板对这些数据进行分析。我们使用的Elastic Stack版本是v5.1.1。

12.2.1 理解Meetup API

Meetup API是多个版本（v1、v2和v3）的组合，其示例请求URL如下所示：

① 译者注：原纸介质图书上未有此图，但电子版上有。这里采用电子版的内容。



- A：这是实际的 Meetup API 主机，一个安全的 URL。
- B：这是 REST 服务方法参数。在前面的例子中，/2/cities 方法为指定的参数传递了城市数据。
- C：包含参数 B 所必需的输入参数以及订单信息或页面大小（结果数量）。例如，在前面的 URL 中，国家参数将用于获取其中的城市，最多返回 200 个结果。

不是所有的方法（method）都可以在没有密钥的条件下使用。如果想要获取有关活动的信息，需要提供 API 密钥进行授权。要获得密钥，应在 meetup.com 上创建账户，然后从以下 URL 获取密钥：

```
https://secure.meetup.com/meetup_api/key/
```

一旦有了密钥，就应将密钥作为一个参数，之后再添加另一个参数，其参数名称为 sign，其值设置为 true。例如获得印度城市 Ahmedabad 的公开活动的网址如下所示：

```
https://api.meetup.com/2/open_events?country=in&city=ahmedabad&key=<your-key
-here>&sign=true
```

请注意，在参数 key 的后面，还添加了 sign = true，这被称为 signed URL。这种 signed URL 的概念在下面给出的链接中提供了解释。

```
https://www.meetup.com/meetup_api/auth/#keysign
```

几个能被插件使用的服务参数如下表所示：

要获取的内容	参　　数
城市	/2/cities
类别	/2/categories
活动	/2/open_events, /2/events
群组	/2/groups
话题	/topics
场所	/2/open_venues, /2/venues

端点用于满足 API 的 v1 和 v2 版本。

12.2.2 搭建 Elasticsearch

通过提供唯一的节点名称、主机名等来设置 Elasticsearch 是一个好习惯。为此，修改配置文件 config/elasticsearch.yml：

```
node.name: es-node
network.host: 192.168.0.102
http.port: 9200
```

对于简单的设置，只保留一个 Elasticsearch 节点。运行以下节点：

```
$ ./bin/elasticsearch
```

Elasticsearch 现在已经准备就绪，应该已经能够正常收到由 Logstash 发来的数据。

12.2.3 准备 Logstash

要从 Meetup 收集数据，我们开发了一个插件。这个插件可以在 GitHub 网站中找到，链接是：

```
https://github.com/kravigupta/mastering-elastic-stack-code-files/tree/5.1.1/
Chapter12/logstash-input-meetupplugin
```

可以下载插件并将其安装到 Logstash 中。如果要安装和测试此插件，请执行以下命令来下载 gem 文件。

```
$ ./bin/logstash-plugin install /path/to/logstash-input-meetupplugin-0.1.
0.gem
```

> TIP 要了解更多信息，可参考第 3 章探索 Logstash 及其组件中插件的命令行操作部分的安装插件中介绍的内容。

我们需要了解如何使用插件以及可能要求的必要配置。该插件的开发用于收集不同城市的数据、Meetup 的活动、场所、群组、话题以及类别。人们可能不愿意手动收集所有的数据，因此插件提供了配置选项，以便于编写 Logstash 配置文件。该插件甚至允许仅提供国家代码，并收集首选城市的数据。我们来看一下可以配置哪些参数。

- countryCode：String 类型，必选项。该参数指定应收集哪些数据的国家代码。这是 ISO 国家代码，最多两个字符。这些代码也可以在 https://countrycode.org/网站获得。这是一个必填字段。如果缺少这部分信息，插件将无法正常工作。

> 该插件不允许同时为多个国家/地区爬取数据。但是,可以提供不同的 countryCode 多次运行 Logstash 来完成数据爬取。

- key:String 类型,必选项。该参数是一个必需的参数,需要访问 Meetup API。可以从本章前面所述的 https://secure.meetup.com/meetup_api/key/中的会议网站获取。
- interval:Integer 类型,可选项,默认值为 900。在收集 Meetup 数据的过程中,该参数表示等待下一次迭代完成的时间间隔。这种设置也确保了 Meetup 不会将我们的操作视为恶意请求,进而封禁我们的访问。此外,Meetup 数据不会频繁变化,因此也可以根据需要,设置更长的间隔时间。
- cityNames:String 类型,可选项,默认值为空。该字段用来指定要收集数据的城市。如果没有指定,插件会自动获取指定国家的 N 个首选城市,其中 N 是 citiesCount 指定的值。

可以使用国家、城市和州的名称来获取 Meetup 数据,state 参数是可选的。如果调用 Meetup API 来获得一个国家的城市,则州名不总是出现在响应中。调用 API 获取城市信息的响应内容详见 https://secure.meetup.com/meetup_api/docs/2/cities/#response。

调用 API,通过城市名称来查找一个州的时候,返回的数据会清楚地显示包含指定城市的州名。在爬取城市数据时,例如获取 Meetup 活动数据,如果响应信息中包含州名,则必须指定州名。因此,在指定要爬取数据的城市名时,应该同时指定州名。可以在指定城市名时使用以下两种格式:

- 对于没有州的国家,请仅使用逗号分隔的城市名称。例如:

```
cityNames: "Ahmedabad,Pune,Bangalore,Chennai"
```

- 对于有州的国家,城市名称也应附加:statecode 参数。例如:

```
cityNames: "New York:NY,Chicago:IL,Washington:DC"
```

  在这里,New York 是城市名称,NY 是州名。同样,IL 是芝加哥市所在的州名,DC 是华盛顿所在的州名。

- citiesCount:Integer 类型,可选项,默认值为 5。仅当未指定 cityNames 时才使用此字段,并且只收集该参数指定数量的城市数据。
- enableTopics:Boolean 类型,可选项,默认值为 false。该参数表示是否应收集话题相关数据。作为参考,话题请求的响应内容详见 https://secure.meetup.com/meetup_api/docs/find/topics/#response。
- enableVenues:Boolean 类型,可选项,默认值为 false。该参数表示是否收集活动场

所相关数据。作为参考，活动场所请求的响应内容详见 https://secure.meetup.com/meetup_api/docs/2/open_venues/#response。

- enableGroups：Boolean 类型，可选项，默认值为 false。该参数表示是否收集 meetup 群组相关的数据。作为参考，群组请求的响应内容详见 https://secure.meetup.com/meetup_api/docs/2/groups/#response。
- enableCategories：Boolean 类型，可选项，默认值为 false。该参数表示是否应收集类别相关的数据。作为参考，类别请求的响应内容详见 https://secure.meetup.com/meetup_api/docs/2/categories/#response。
- enableMeetup：Boolean 类型，可选项，默认值为 true。该参数表示是否收集 meetups(events)相关的数据。Meetup 活动请求的完整响应内容详见 https://secure.meetup.com/meetup_api/docs/2/open_events/#response。默认情况下，该字段设置为 true；这是插件的核心。

因为要收集多个国家的数据，因此需要在配置中不断更改 countryCode 的值，并逐次运行 Logstash。

这个配置将为插件提供 API 密钥和 10 分钟的时间间隔。

如果仅为少数城市爬取数据，则对美国的配置如下所示：

```
input { meetupplugin{
    countryCode => "US"
    cityNames => "Chicago:IL,Washington:DC,Houston:TX,NewYork:NY"
    ...
} }
```

对于活动场所、群组和聚会中的所有项目，每个城市只需要一个 API 即可调用。因此，为了获得首选的 10 座城市的数据，程序中将进行 30 次 API 调用。因为 meetup.com 每小时只允许 200 次调用，所以需要进行适当的规划，以免用尽 API 调用次数，收到报错信息。

每个群组都有一些相关的话题，总计其数量可以达到几百个。每个话题都可以提供成员数量的信息，这对我们的规划而言是一个很好的衡量标准。然而，每个主题需要调用一个 API，这是不可行的，因此在前面的配置中，话题的部分被禁用。

群组、聚会和活动场所等位置信息具有相关的纬度和经度，可以用这样的信息在地图上显示数据。需要进行调整，才能画出一幅地图。纬度和经度转换为位置对象已经在插件中提供，只需要使用过滤器来处理其他工作。过滤器配置如下所示：

```
filter {
    mutate {
        add_field => ["[geoip][location][lat]","%{[location[lat]}"]
```

```
        add_field => ["[geoip][location][lon]","%{[location[lon]}"]
    }
}
```

在这里，使用 mutate 过滤器并添加了字段 geoip，在位置字段中选择一些字段，这些字段已经被添加到插件的活动对象中。

除了这个配置，还需要指定 Elasticsearch 的输出：

```
output {
    elasticsearch {
        hosts => "localhost:9200"
        index => "meetup"
        document_type => "meetup_data"
        doc_as_upsert => true
        document_id => "%{document_id}"
        template => "/Users/ravi.gupta/templates/meetup-template.json"
        template_name => "meetup-template"
        template_overwrite => "true"
    }
}
```

请注意，已将 doc_as_upsert 指定为 true，并指定了 document_id 字段。Meetups 不是日志事件，我们需要做的是更新现有的聚会、活动场所、群组或任何其他数据。

该插件使用 document_id 字段名来存储文档的唯一 ID。要强制 Elasticsearch 更新文档的 ID 字段，需要使用 document_id 字段，并提供插件使用的 ID 字段，这在配置中指定为%{document_id}。

参数 geoip 需要三行配置来作为模板。只需要将模板内容复制到某个位置，并提供这个模板。模板的示例内容可以在如下地址找到：logstash-5.1.1/vendor/bundle/jruby/1.9/gems/logstash-output-elasticsearch-5.4.0-java/lib/logstash/outputs/elasticsearch/elasticsearch-templatees5x.json。

复制这个文件，放在一个位置并重命名，例如 meetup-template.json。打开文件并将模板字段的值从 logstash* 更改为 meetup*，这是我们的索引模式。

执行以下命令启动 Logstash：

```
$ ./bin/logstash -f conf/meetup.conf
```

现在，Logstash 将开始从 meetup.com 中获取数据，并存储在 Elasticsearch 的索引中。之后就可以在 Kibana 中可视化这些数据。

> 请注意，要获取多个国家的数据，请更改配置中的 countryCode 并再次运行 Logstash，以此对每个国家重复一次。数据越多，可视化和分析的效果就越好。

12.2.4 搭建 Kibana

在配置 Kibana 时，需要在 config/kibana.yml 配置文件中指定 Elasticsearch 运行所占用的端口，如下所示：

```
elasticsearch.url: "http://192.168.0.102:9200"
server.name: "Kibana-node-stack"
```

Kibana 一经搭建和配置，即可开始对传入的数据执行可视化。在 Kibana 的安装目录中运行以下命令：

```
$ ./bin/kibana
```

此处有一个需要理解的地方——插件添加了一个附加字段来标识文档的类型，该字段是 meetup_data_type，它被分配到以下参数之一：

- meetup：Meetup 活动；
- group：Meetup 群组；
- venue：Meetup 活动场所；
- topic：Meetup 话题；
- category：Meetup 类别。

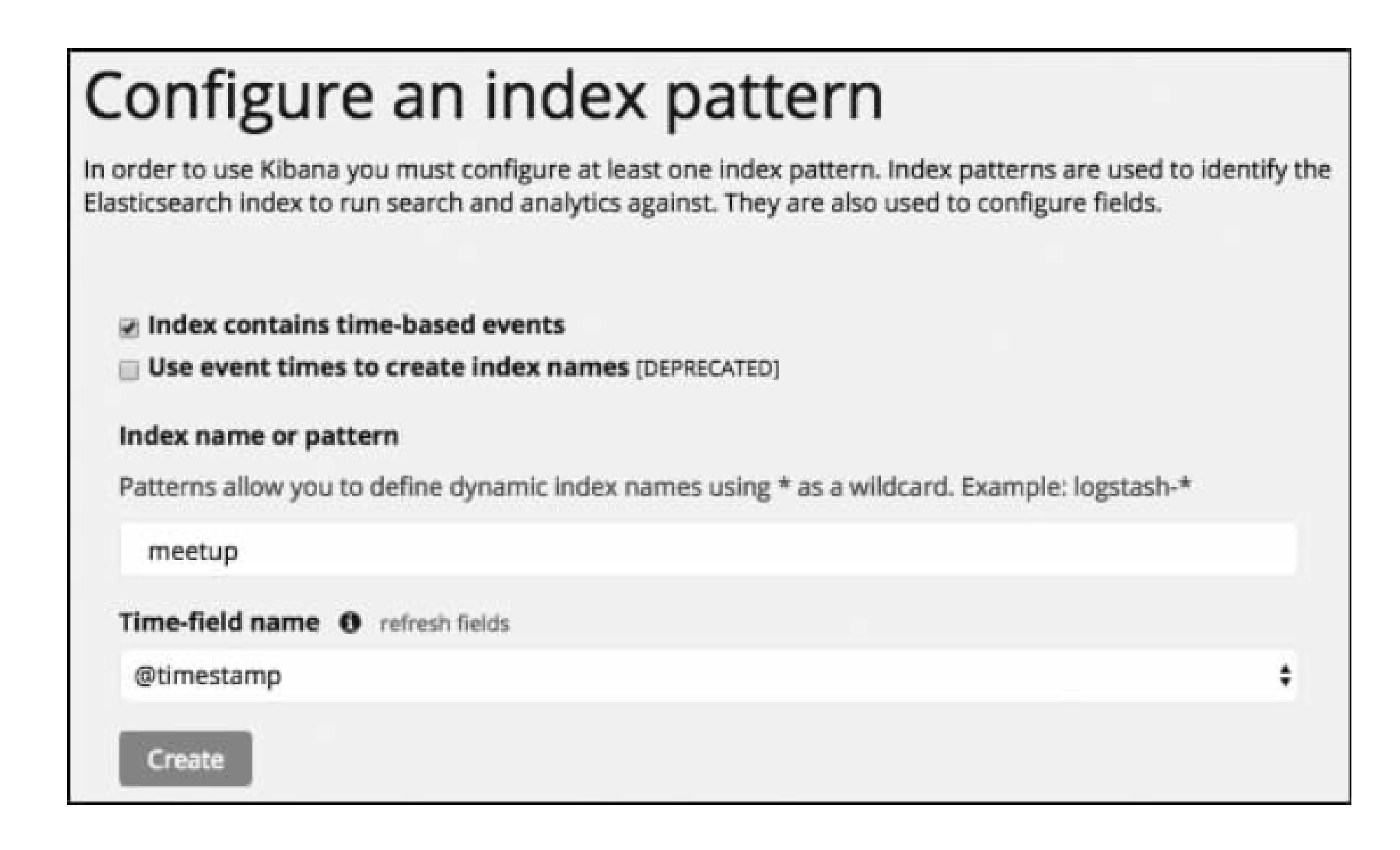
将 Kibana 中的过滤器应用于数据来执行可视化时，这些字段将非常有用。

12.3 使用 Kibana 分析数据

当 Kibana 启动并运行时，即可开始可视化 Meetup 数据。使用 Kibana 之前，需要配置其中的索引。在 Elasticsearch 中的索引名称是 meetup，是在 Logstash 的输出配置中指定的，现在就来设置 Kibana 中的索引模式。转到 **Kibana | Management | Index Patterns | Configure an Index Pattern**（**Kibana**|**管理**|**索引模式**|**配置索引模式**），或单击 **Add New** 来添加新索引。添加该索引模式，并将@timestamp 字段指定为时间字段名称。

> 也可以使用 * 作为通配符，并以 meetup * 来匹配相关的索引。因为只有一个索引，并且知道它是 meetup，所以不需要使用通配符。这也有助于保持数据的准确性，以防与其他具有相同模式的索引混淆，例如 meetup-india、meetup-us 等。

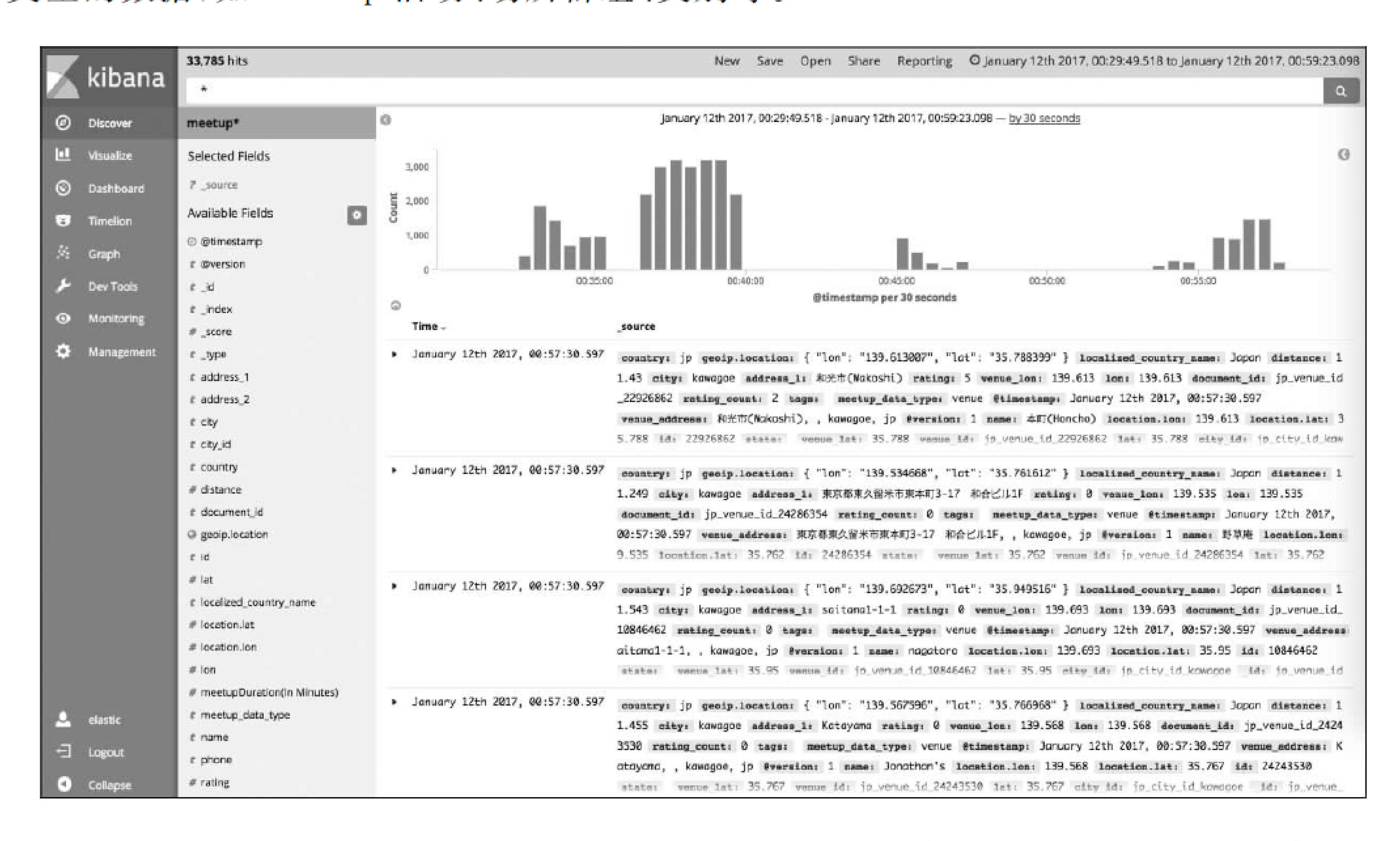
单击**创建(Create)**,该索引的索引模式将以下图所示的方式被创建:



完成此操作将添加索引模式,索引 meetup 中的所有字段 field 将与其类型 type 一起列出。

12.3.1　内容过滤

索引模式一经设置,Discover 页面将铺满各类事件信息。默认情况下,该页面将显示所有类型的数据,如 Meetup 活动、场所群组、类别等。



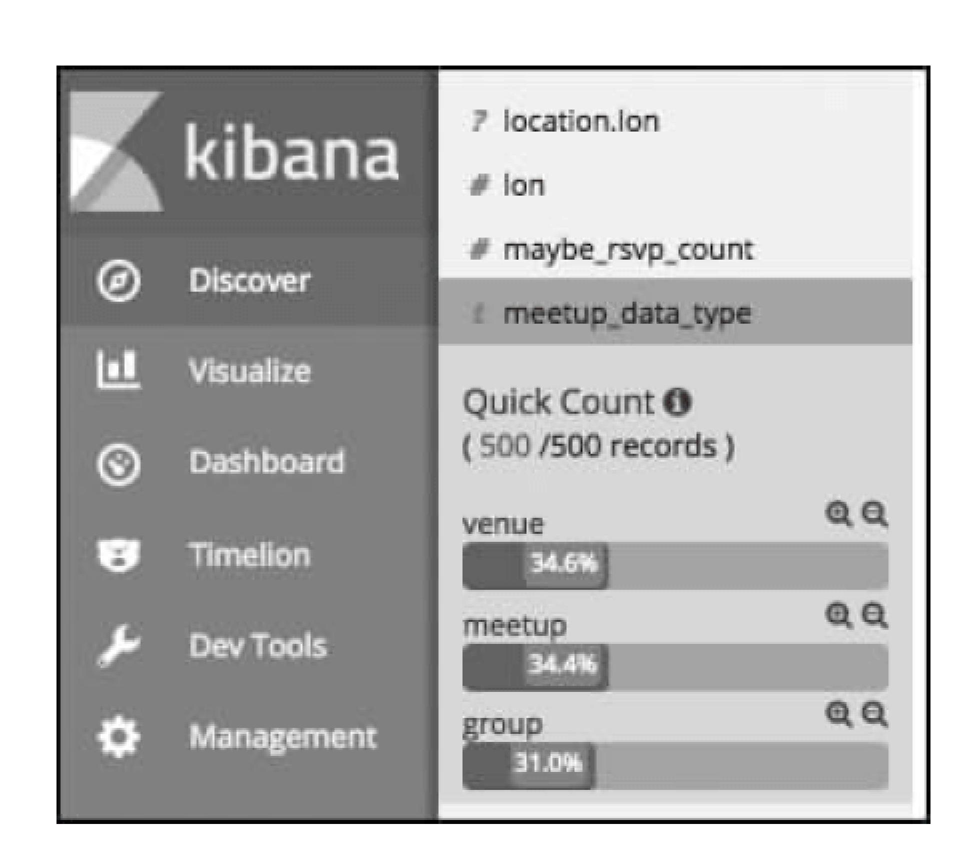
从上图中可以看到有很多字段出现在 Discover 界面左侧的面板中,这些字段都是数据中存在的唯一字段。可以使用字段的值来完成数据搜索、快速过滤及检查字段是否存在等

操作。

在这种界面中,一个字段可以公平地出现在多种类型的数据中。例如,名称的信息可以出现在 Meetup 活动、场所和群组中。还有其他字段也是这样,如纬度(lat)和经度(lon)。如果想在地图上绘制 Meetup 活动的分布情况,那么只需要识别与 Meetup 活动相关的信息。在这种情况下,meetup_data_type 字段将是有帮助的,可以使用过滤器将其中一种类型的数据清楚地分离出来。大多数可视化应用仅使用一种数据类型,例如,Meetup 事件相关的可视化仅使用包含值 meetup 的 meetup_data_type 字段。同样,meetup_data_type 也将被设置为活动场所。

要添加过滤器,可以参考下述步骤进行可视化处理:

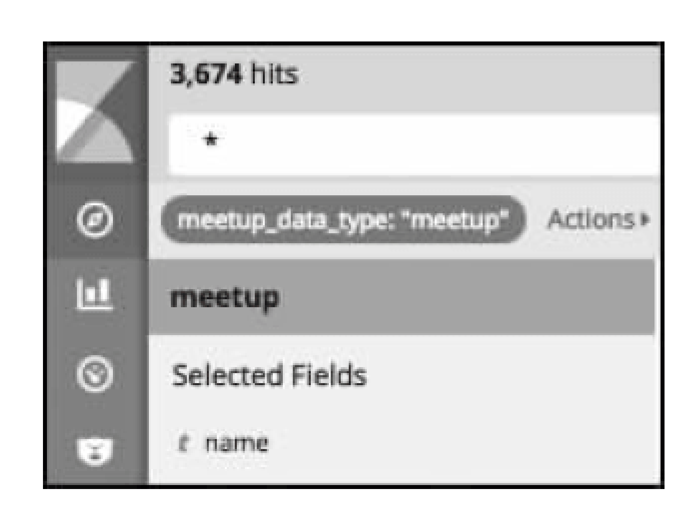
(1) 将在 meetup_data_type 字段上使用过滤器,并在此可视化中设置其值为 meetup。转到 **Kibana** | **Discover** 界面,并在字段列表中选择字段 meetup_data_type,然后选择 meetup。



(2) 选择带有加号(+)的放大镜图标,这将为查询添加一个过滤器。而在程序后台,执行的则是如下所示的代码:

```
{
    "query": {
        "match": {
            "meetup_data_type": {
                "query": "meetup",
                "type": "phrase"
            }
        }
    }
}
```

(3) 搜索框下方会添加一个过滤器，其中注明了 meetup_data_type:"meetup"。



(4) 在界面中固定此过滤器，以便可以直接使用它来进行可视化：



> 固定过滤器是十分有用的，这样可以便于访问 Kibana 的任何一个标签，这个过滤器始终可用。

该过滤器将被应用于所有可视化内容，并且会相应地对过滤器进行更改。

现在已经准备好，可以开始对数据执行可视化了。我们将在人气、计数、加入模式等方面对活动、群组、场所、类别执行可视化。从国家的 meetup 使用量开始，我们来看看可视化的效果。

12.3.2 按国家统计 Meetup 使用量

假设已经为多个国家运行了 Logstash，现在为此创建一个可视化，执行以下步骤：

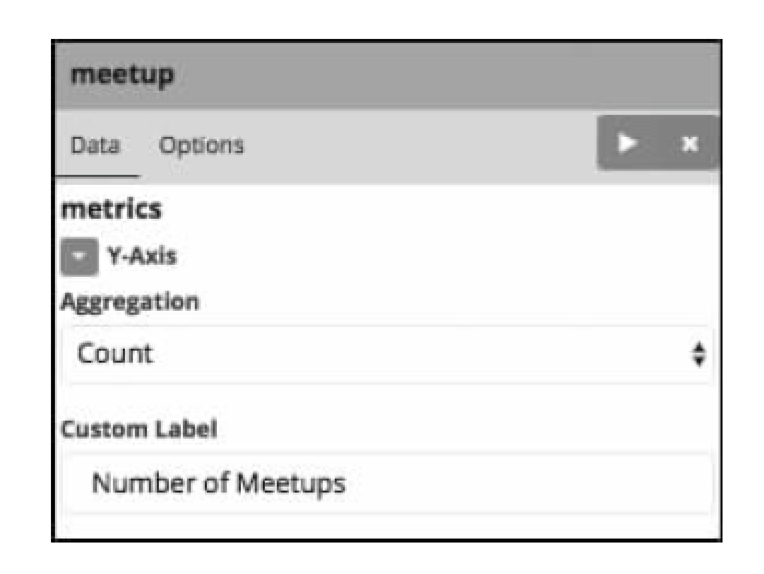
(1) 在内容过滤(Filtering Content)部分中指定 meetup_data_type: 'meetup'。

(2) 转到 Kibana | Visualize | New(位于顶栏)，并选择直方图(Vertical bar chart)。

(3) 在"选择新建的索引"(From a New Search, Select Index，如下图所示)部分中选择已配置的索引为 meetup。



(4) 添加 metrics 指标，使用它可以绘制出条形图。Y 轴已设置为计数聚合，将其保留原样，并为 Y 轴添加自定义标签(Number of meetups，如下图所示)作为 meetup 使用量。



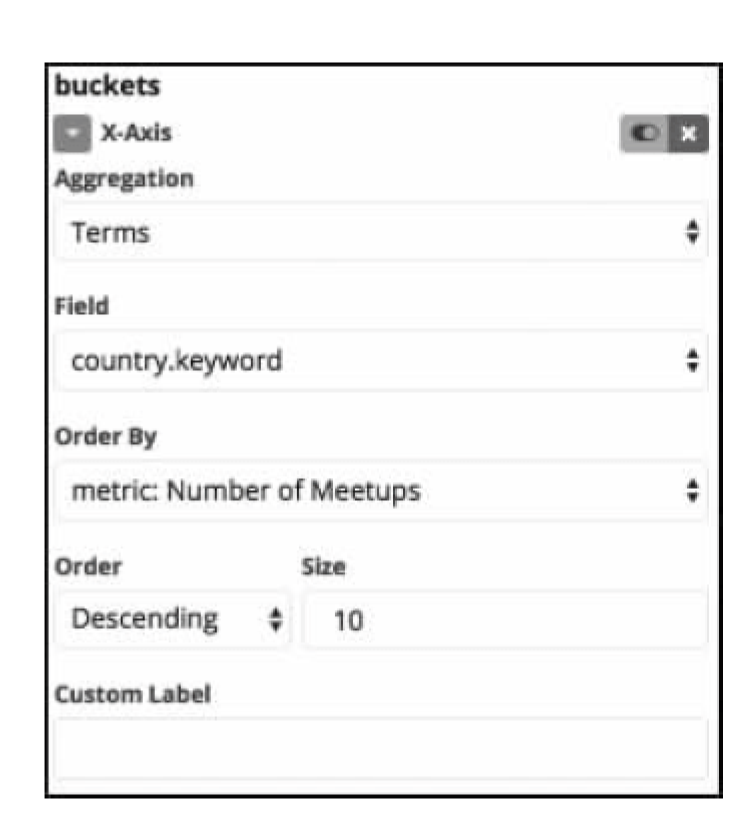
（5）使用 metrics 下方的 buckets 部分将 bucket 添加到表中，从 bucket 列表中选择 X 轴。

（6）选择 Aggregation(聚合)类型为词项(Terms，如下图所示)。

（7）选择字段 Field 为“country. keyword”(如下图所示)。

（8）由于可能有五个以上的国家，所以需要将大小设置为 5 以上。这里设置为 10。

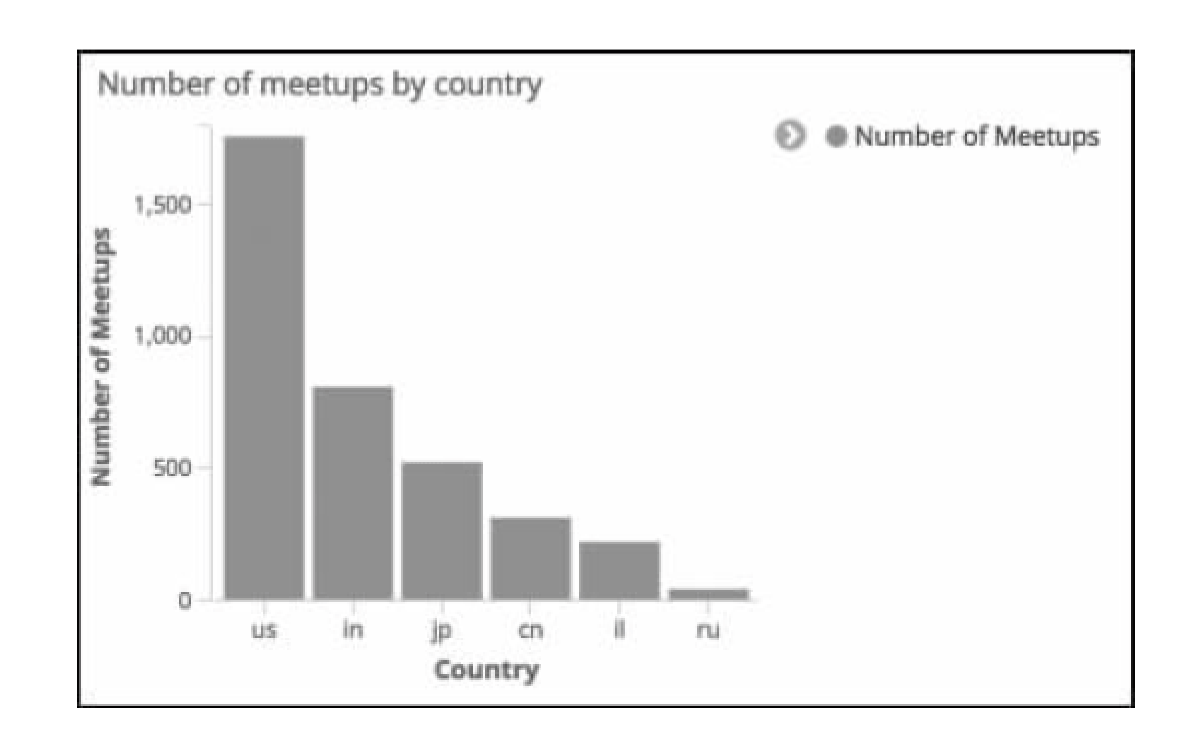
（9）X 轴 buckets 如下图所示。



（10）现在已完成图表的所有设置，单击索引名称下面的 Data 和 Option 选项卡旁边的“Apply Changes”(应用更改)图标，它是如下图所示的播放按钮。



（11）可视化内容将类似于如下图所示。



从该图中可以看到国家代码所代表的不同国家的数据。

12.3.3 世界前10座使用Meetup的城市

前面的可视化是在国家级别执行的，有时可能需要进一步查看首选城市以及Meetup活动活跃的地方的详细信息，而这种可视化仅适用于有数据的国家。如果有所有国家的数据，这将是一个准确而完善的可视化。下面分几个步骤修改可视化设置，来获取城市而不是国家的数据：

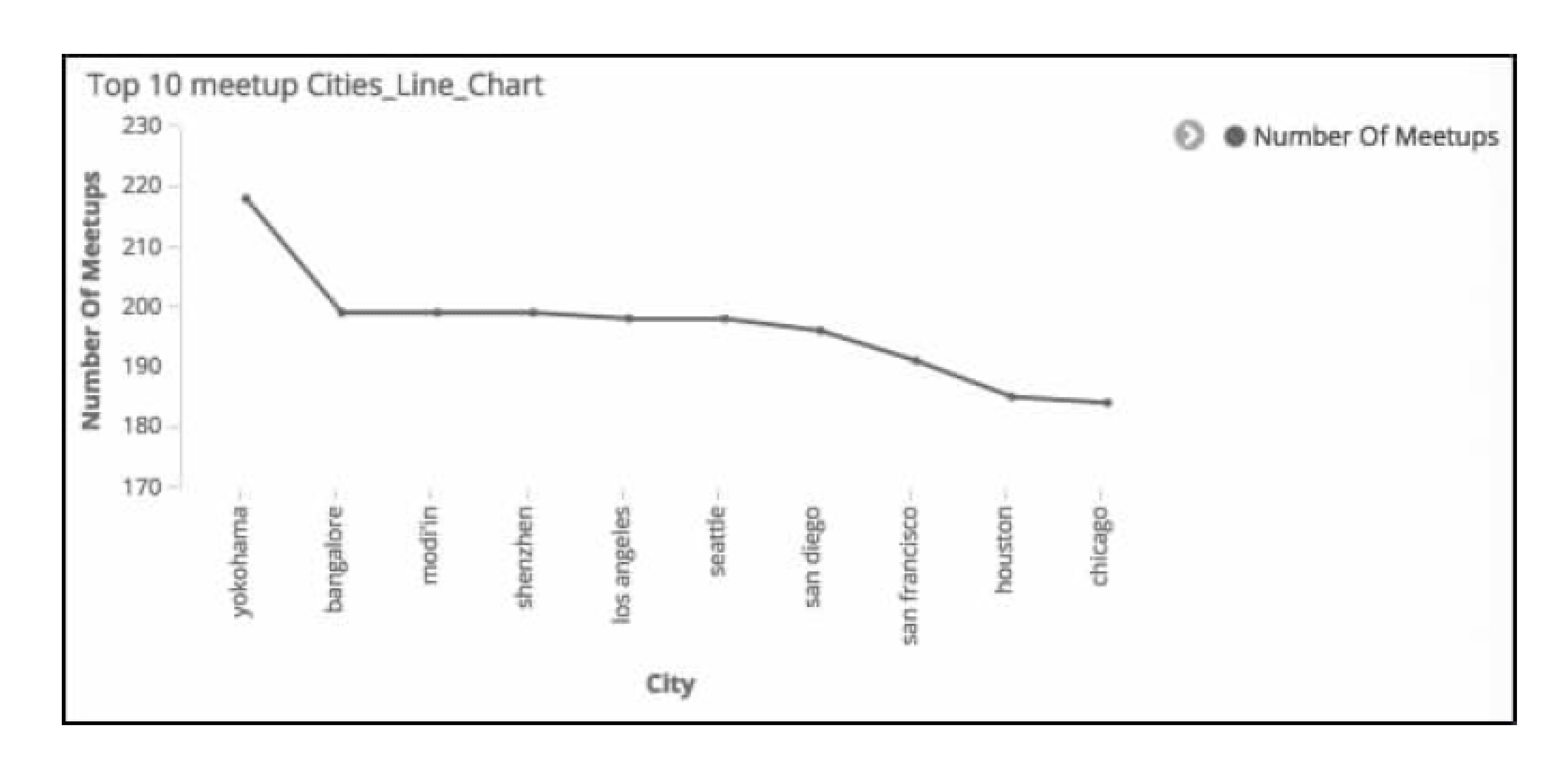
(1) 按照“内容过滤”部分中的步骤来设置 meetup_data_type：'meetup'。

(2) 试试另一种可视化。转到 Kibana | Visualize | New(位于顶栏)| 选择 Line Chart (折线图)，选择索引模式为 meetup。

(3) 将Y轴指标设为Count(计数)，并将其标注为“Number of Meetups”(Meetups 次数)。

(4) 对于X轴上的 buckets，设置 Aggregation(聚合)的方式为 Terms(词项)，选择字段 city.keyword，并将 Size 设置为10。

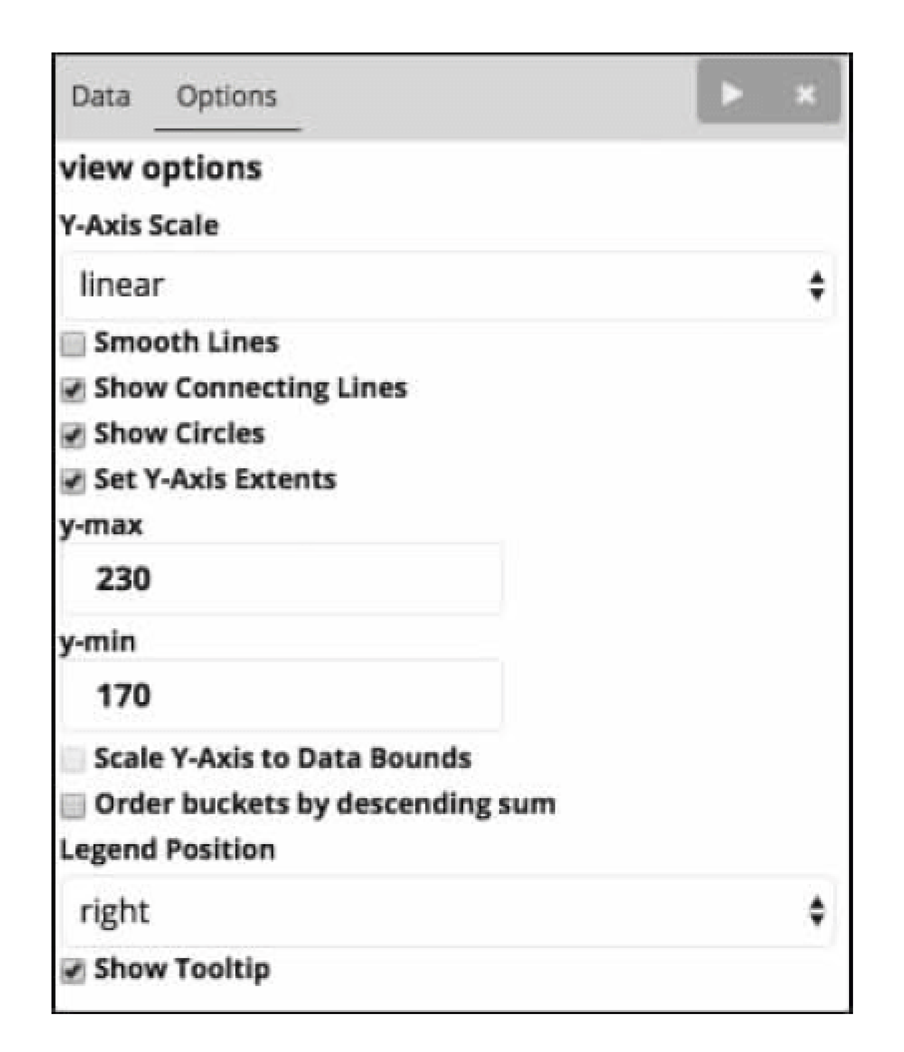
(5) 将此 bucket 的标签设为 City 并应用或按 Enter 键，此时可以看到当时首选城市中 meetup 活动的数量统计，如下图所示。



从该可视化统计图中可以看出，Yokohama(横滨)举办 meetup 活动的数量是最多的。

如果稍加注意，就会发现折线图中的Y轴的数值不以零开始，而是以170开始。可以使用可视化选项卡中的任何选项来设置这些内容。在 Data(数据)选项卡中可以对指标和 bucket 进行设置，而 Options(设置)选项卡可对图表进行更多操作。对于不同类型的统计图表，在 Options 选项卡中提供的设置项不尽相同。在这里的折线图设置中，可以看到设置Y轴范围的选项，如下图所示。

请注意，对于Y轴的范围，已将最小值设置为170，最大值设为230。该值是基于对数据的观察设置的，即任何一座城市中 meetup 活动的最大数量不超过230，最小数量不小于170。



12.3.4 按持续时间分析 Meetup 发展趋势

这个分析可以帮助我们了解大多数聚会的持续时间。这是一个重要的指标，它清楚地显示了 Meetup 活动理想的持续时间，该可视化操作将使用持续时间字段中的值。默认情况下，持续时间字段以毫秒为单位，应该将其转换为分钟或小时，以获得更好的可读格式。可以使用脚本字段(scripted fields)来做到这一点，添加脚本字段的步骤如下所示：

(1) 转到 Kibana | Management | Index Patterns | Scripted Fields | Add Scripted Fields。

(2) 将字段命名为 meetupDuration(In Minutes)。

(3) 使用 Painless 脚本语言。要了解更多关于 Painless 脚本的信息，请参阅第 8 章 Elasticsearch API。

(4) 将以下脚本添加到脚本字段中：

```
doc['duration'].value/(1000 * 60)
```

这将获取索引文档中持续时间字段的值，并将其除以(1000 * 60)，将毫秒转换为分钟。

(5) 将其他所有内容保留为默认值，然后选择“Create Field”(创建字段)。

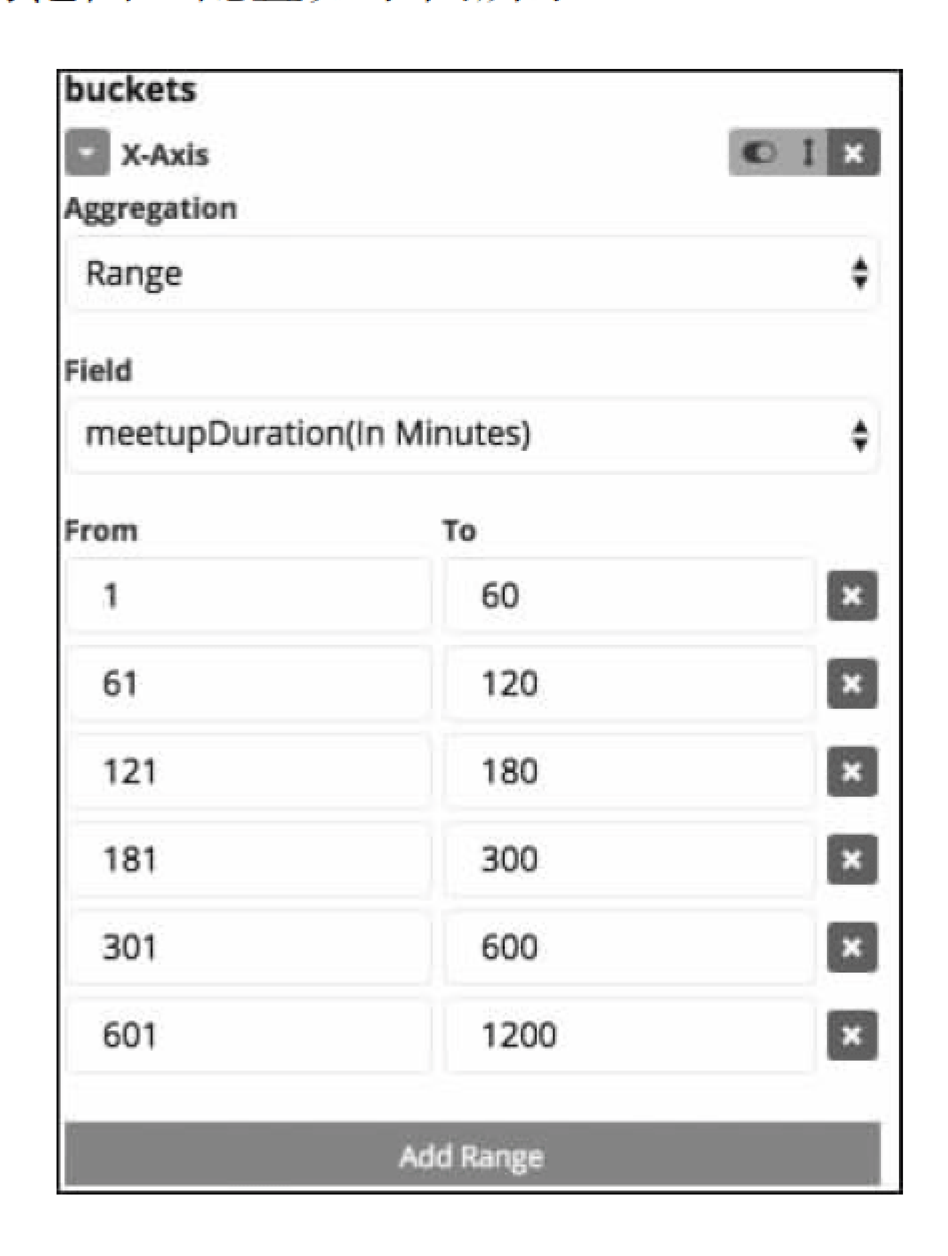
按照以下步骤使用该字段进行可视化：

(1) 按照“内容过滤”部分中的步骤来设置 meetup_data_type: 'meetup'。

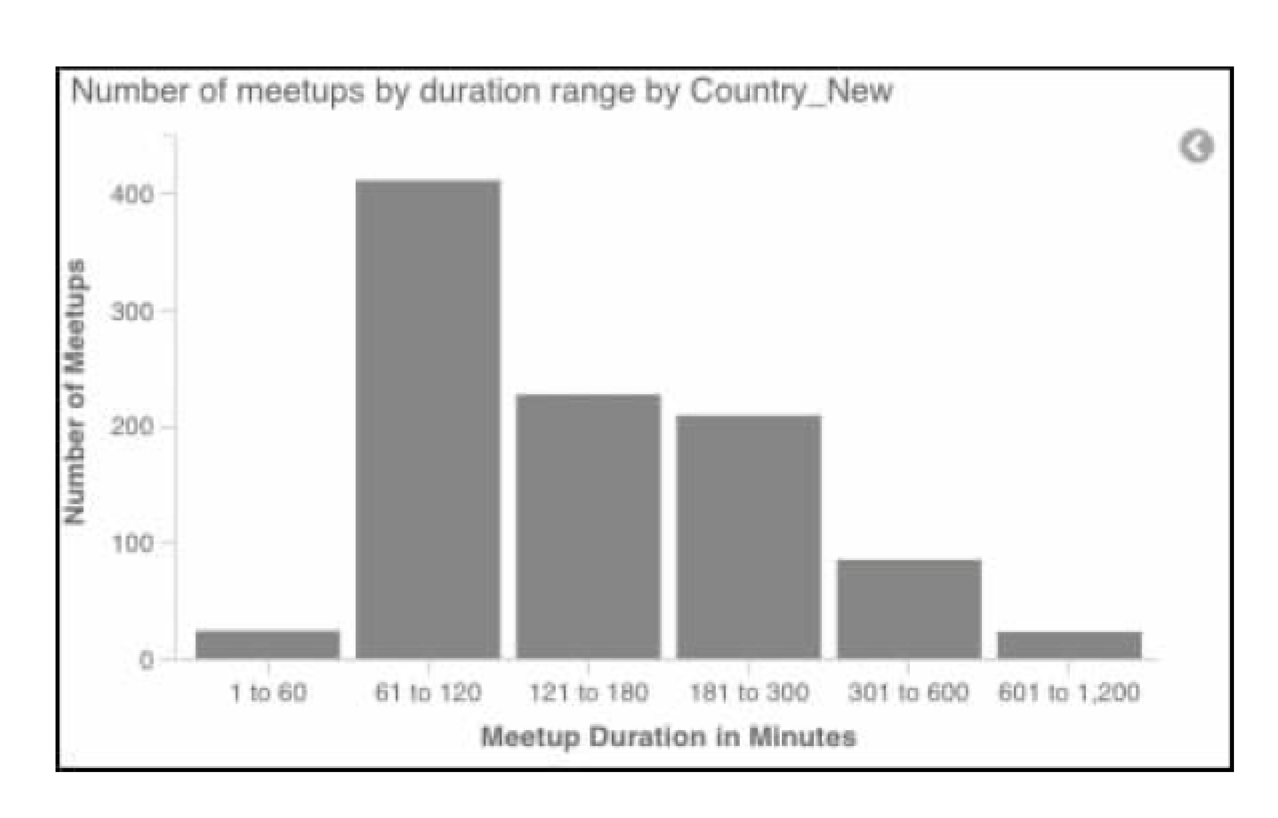
(2) 转到 Kibana | Visualize | New(位于顶栏) | 选择直方图(Vertical bar chart)。选择索引模式为 meetup。

(3) 将 Y 轴的指标设置为 Count(计数),并将其标注为"Number of Meetups"。

(4) 对于 X 轴上的 buckets,设置聚合的方式为 Range(范围)。选择字段 meetupDuration (In Minutes),并添加适当的范围。配置如下图所示。



(5) 应用所做的上述更改,可以看到 Meetup 活动在所选持续时间范围内的数量分布,如下图所示。

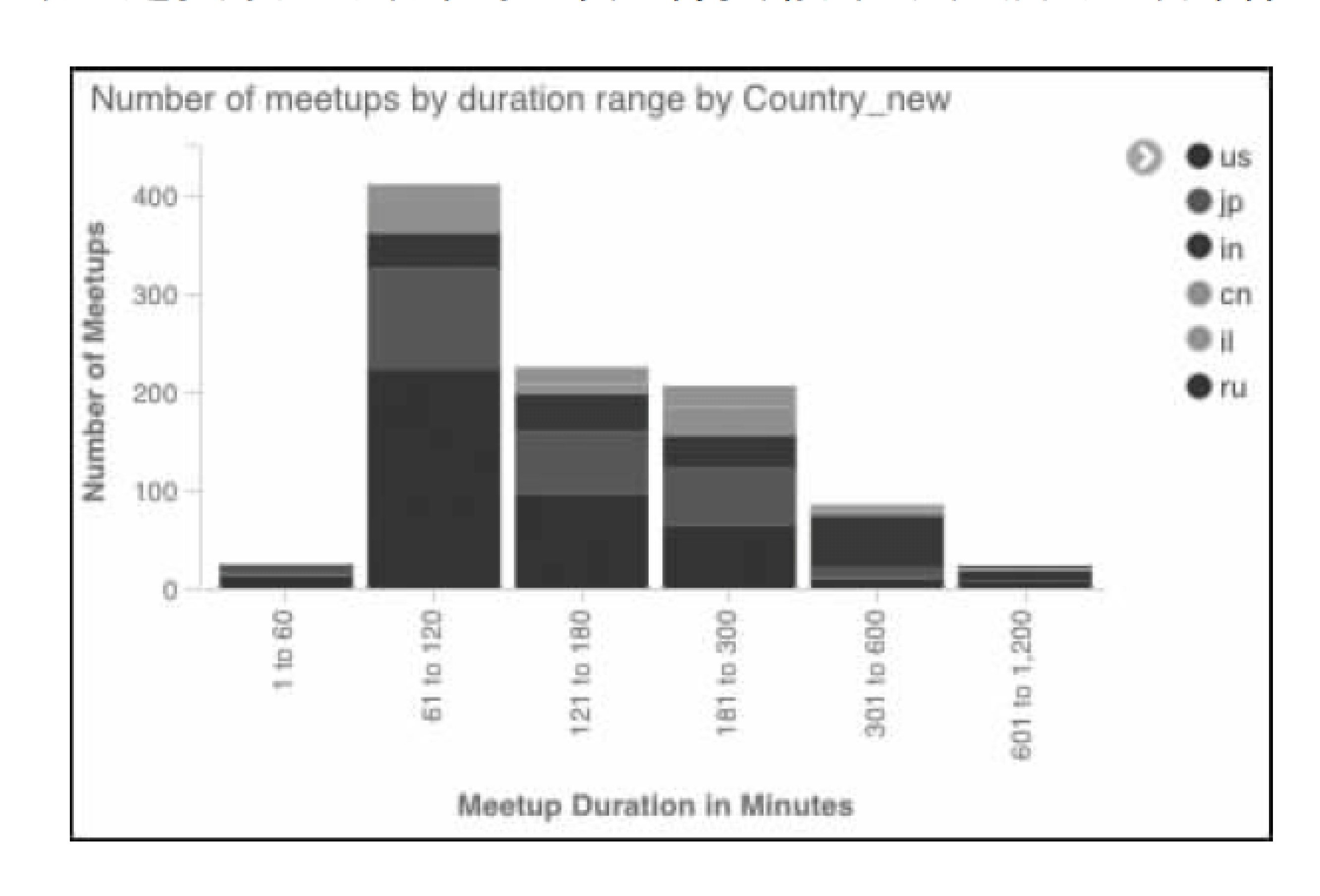


从该可视化统计图中可以看出,大部分 Meetup 活动都是 61～120 分钟的时长,绝大多数 Meetup 活动时长在 1～3 个小时的范围内。

(6) 进一步修改每个国家的统计图。下面添加"Split Bars"类型的子 bucket。在子聚合(Sub Aggregation)中选择词项(Terms),并将其字段设置为 country. keyword。配置应如下图所示。

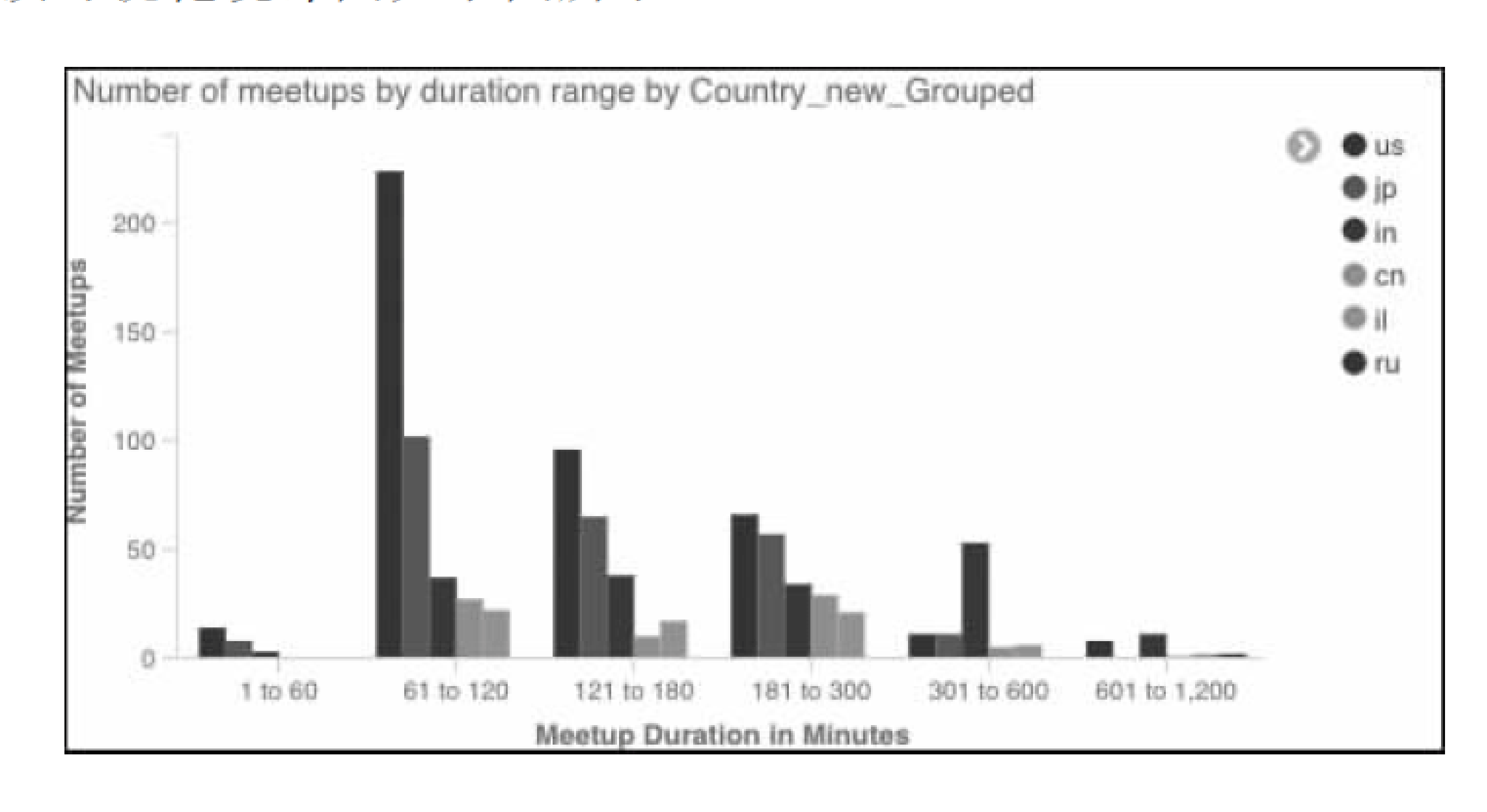


（7）应用所做的上述更改，可视化统计图将类似于下图所示的内容。



可以看到，现在每个国家的 Meetup 活动数量统计都相互叠加了。

（8）如果要将堆叠的条更改为分组，可以将选项中条形模式的值更改为 grouped（分组）。更改后，该可视化统计图如下图所示。



从第一个持续时间范围可视化统计图中不难看出，世界各地的大多数 Meetup 活动都不到 3 小时。现在，可以从最后一张统计图中不同国家的 Meetup 活动数据中看出，在印度，大多数 Meetup 活动时间都超过了 3 小时。

12.3.5 按 RSVP 计数统计 Meetup 使用量

类似于 Meetup 活动的持续时间，可以按以下步骤，对这种范围使用 RSVP① 计数，并查看 Meetup 活动完成了多少个 RSVP：

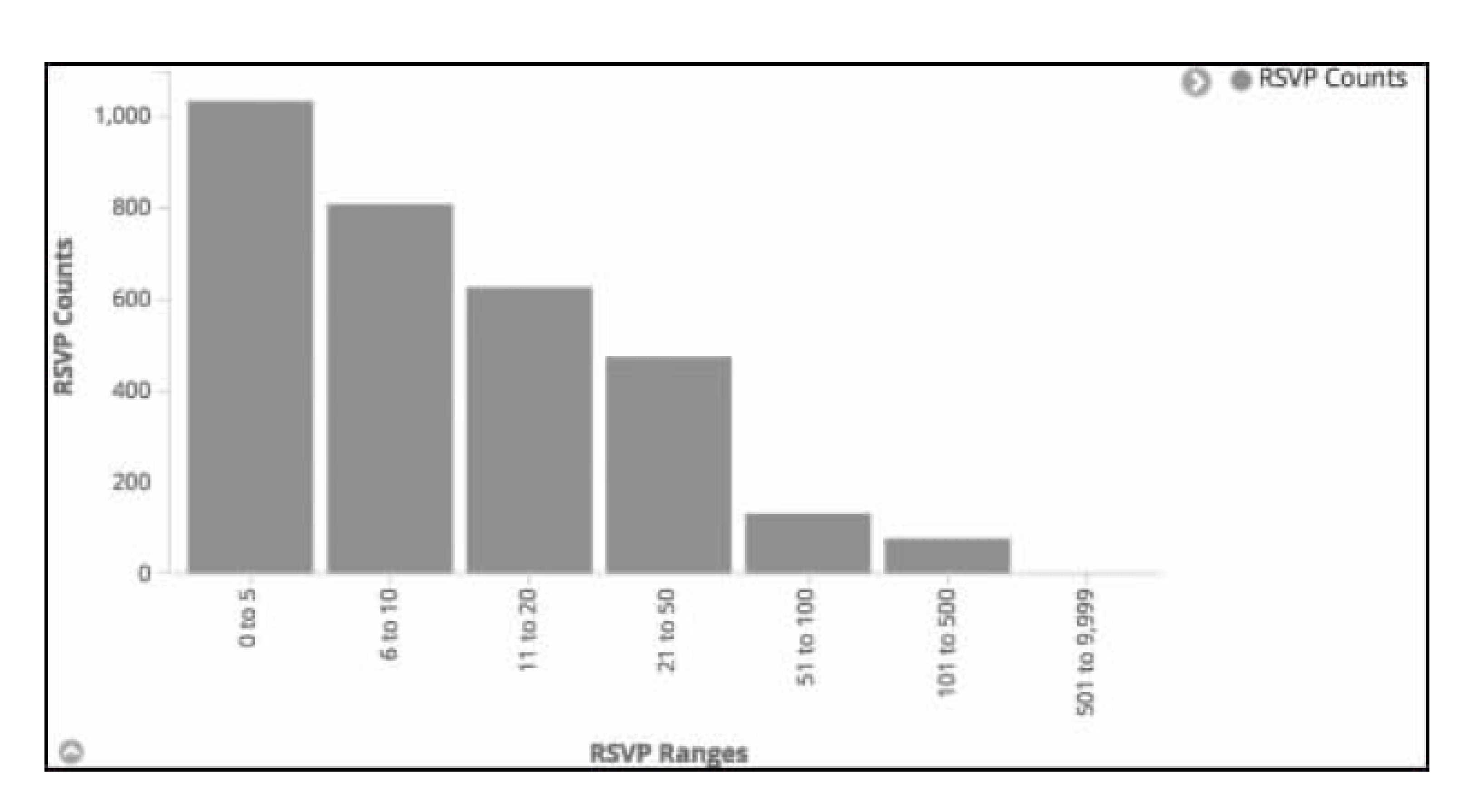
（1）按照"内容过滤"部分中的步骤来设置 meetup_data_type：'meetup'。

（2）转到 Kibana|Visualize|New（位于顶栏），选择直方图（Vertical bar chart），选择索引模式为 meetup。

（3）将 Y 轴指标设为 Count（计数），并将其标签设为"Number of Meetups"（聚会次数）。

（4）对于 X 轴上的 buckets，设置聚合的方式为 Range（范围），接着选择字段 yes_rsvp_count，并添加适当的范围。可以设置从一个正数到 500 及以上的各种范围。

（5）单击应用更改，yes RSVP 数据（即成功邀请数量）将按照定义好的范围绘制出来，如下图所示。



可以看到，大多数 Meetup 活动邀请都会收到少于 10 个回复，此数据仅适用于即将举办的 Meetup 活动。与以前的可视化内容类似，这里也可以进一步划分每个国家的数据。

① 译者注：R.S.V.P 是法语 Répondez s'il vous plait 的缩写，表示活动参与者接到邀请后，无论能否出席均应尽快回复。

12.3.6 国家分组统计

查看可用数据中每个国家的群组数量统计，可以通过执行以下步骤来完成：

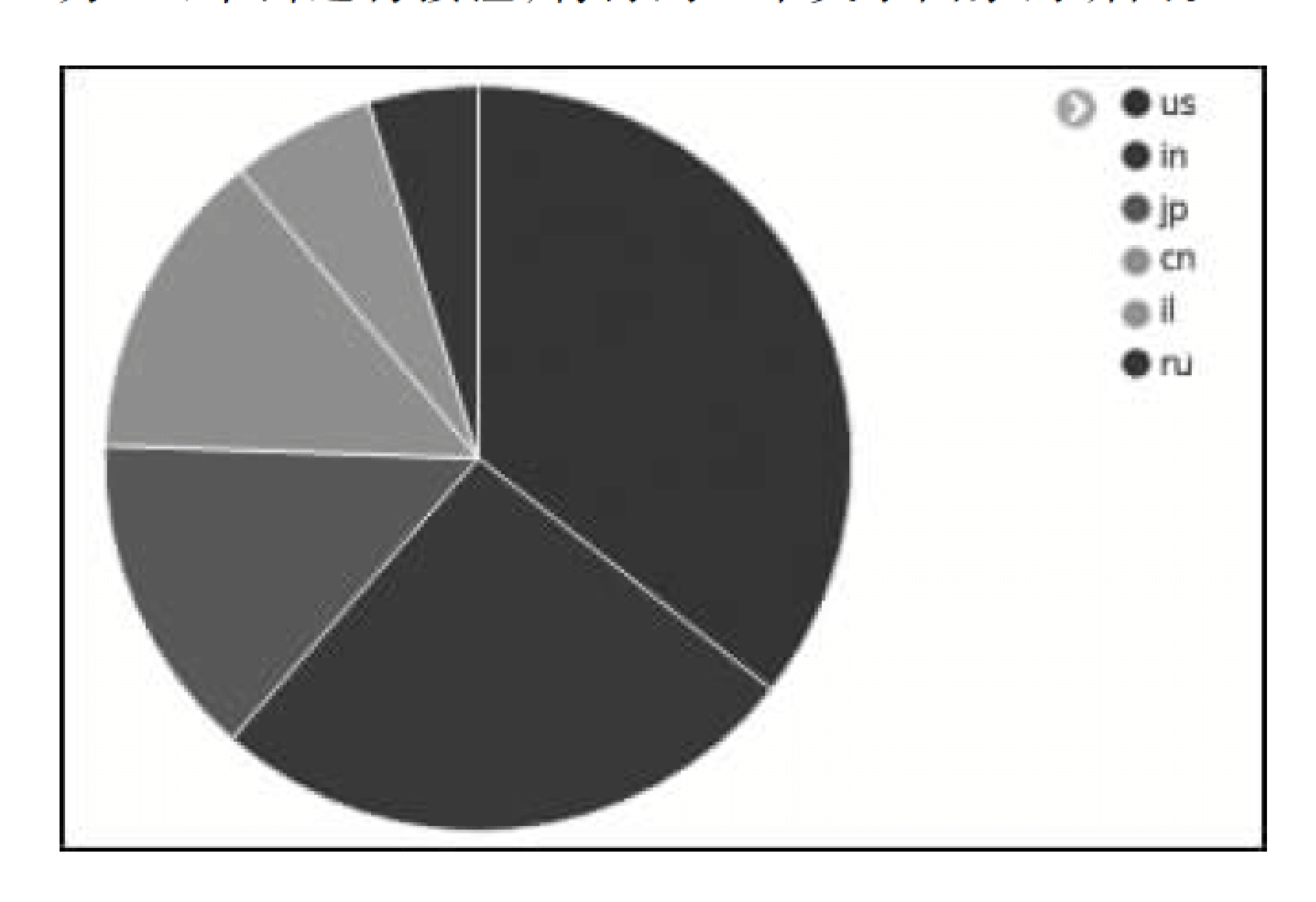
（1）按照“内容过滤”部分中的步骤来设置 meetup_data_type：'group'。

（2）转到 Kibana|Visualize|New（位于顶栏）|，选择饼图（Pie chart），选择索引模式为 meetup。

（3）将其中的 metrics（指标）和 Slice size（层数）的聚合方式设置为 Count（计数），并将其标注为“Number of groups”（群组数量）。

（4）在 buckets 中添加 Split Slices（拆分）的聚合，方式设置为 Terms（词项），并将选择字段 country.keyword。

（5）设置 Size 为 10，单击运行按钮；将得到一个关于国家的饼图。



12.3.7 加入群组的模式统计

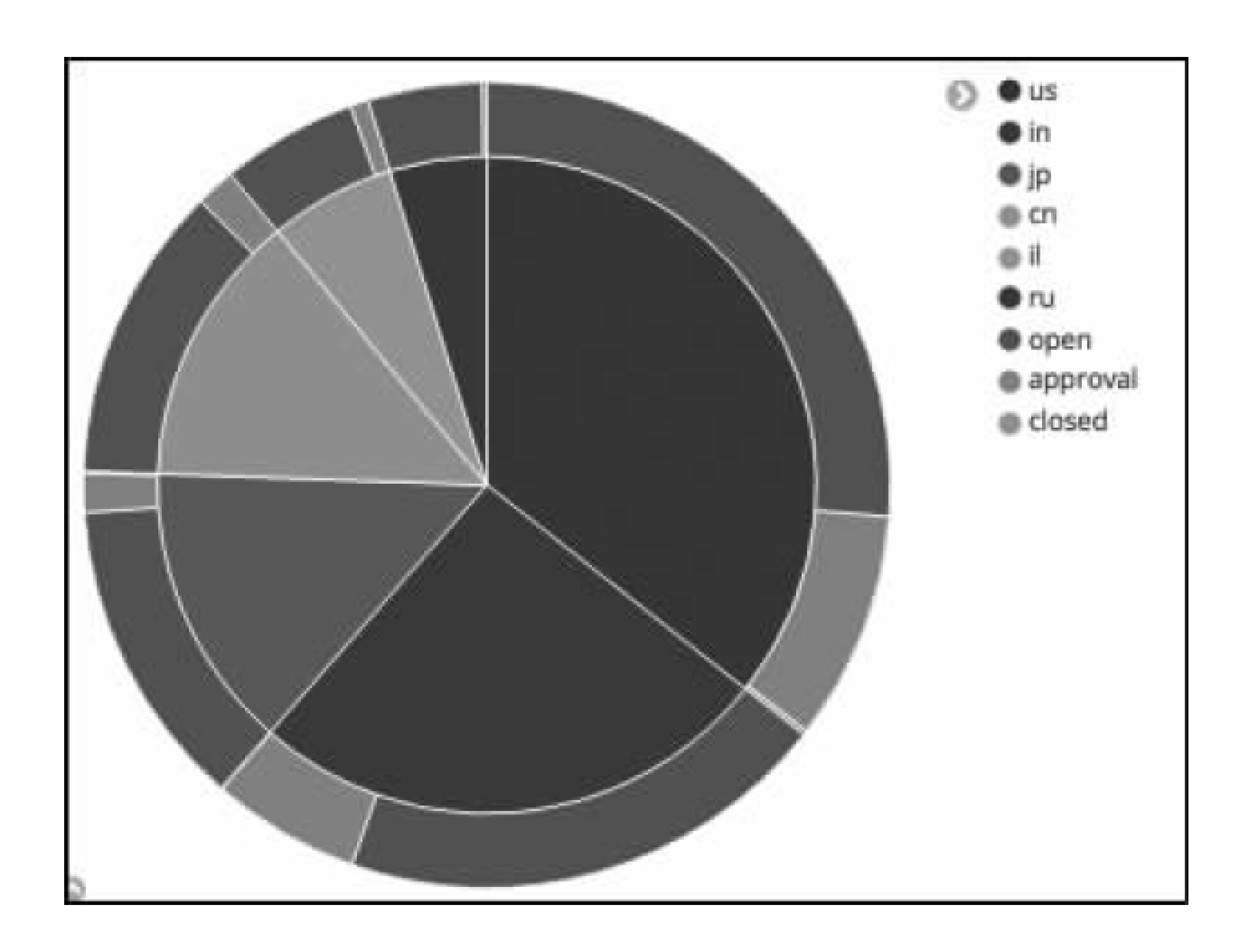
可以进一步分析基于加入模式的群组数量，加入模式可以显示出群组或国家中社交的开放或限制程度。以下是加入模式的三种类型：

- 开放：可以找到并加入这些群组，并且不需要批准。
- 需要批准：该模式允许查找和加入群组，但成员资格需要群组管理员批准。
- 封闭：只能由管理员添加。

再向这个以国家分组的饼图中添加一个外层，这可以通过执行以下步骤来实现：

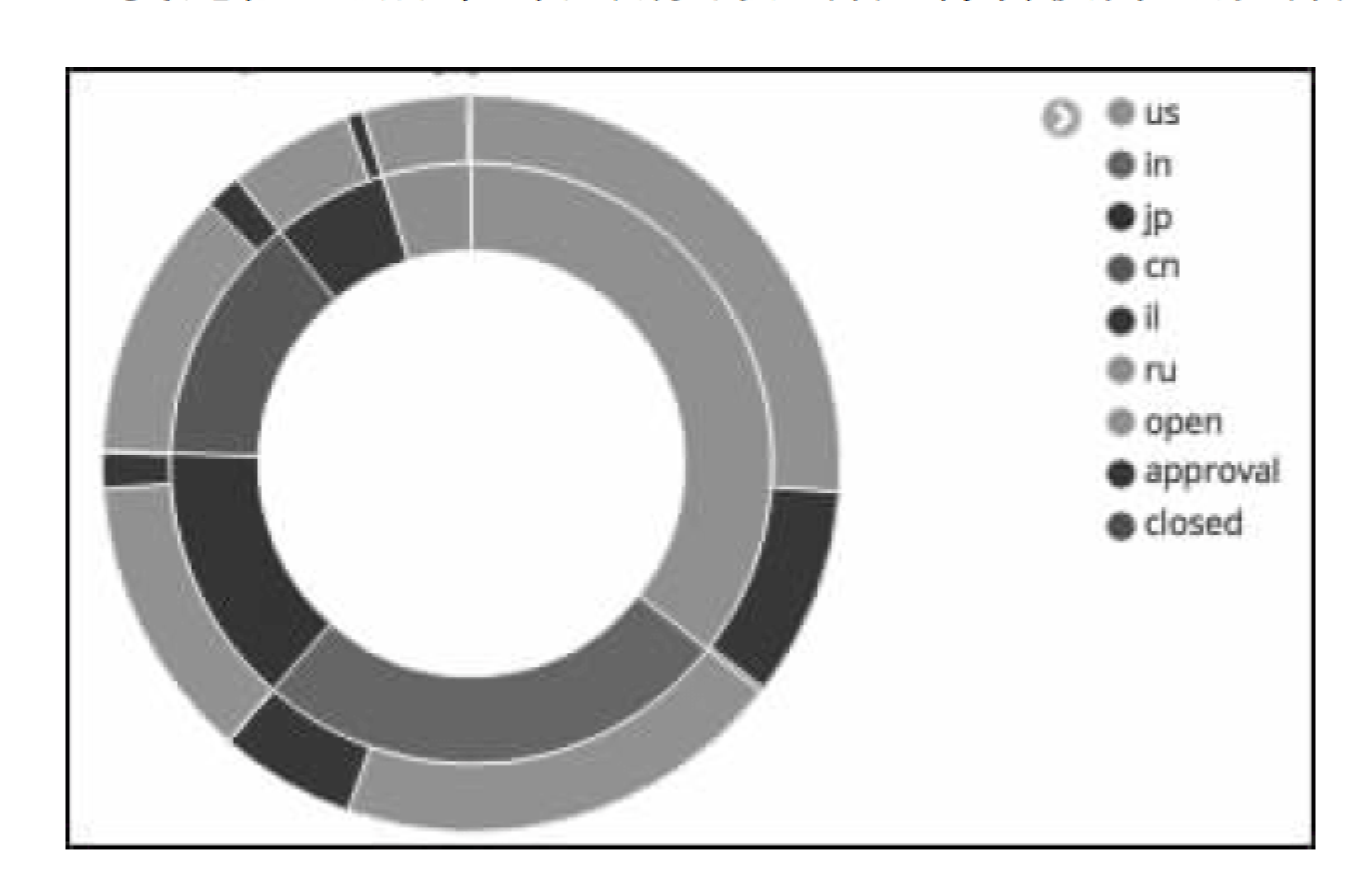
（1）在 buckets 中再添加一个 Split slices 聚合，方式设置为 Terms，并设置字段为 join_mode.keyword。

（2）单击应用更改，可以看到更新后的饼图如下图所示。



从上面的图表中可以看出,大部分 Meetup 群组都是开放的,这是一件好事。它可以让你立即加入群组,并进一步加入群组举办的 Meetup 活动。

(3) 如果想为图表添加不同的风格(不只是饼图),也可以添加环形饼图。打开 Options 选项卡,并选中 Donut 复选框,之后单击应用更改,饼图将转换为环形饼图,如下图所示。



12.3.8 热门类别

所有的 Meetup 群组都与某些类别有关。根据我们的数据,可以找到最受欢迎的热门类别。如果有兴趣了解这些类别下 Meetup 群组的确切数量,可以使用数据表(Data table)。生成类别表格的步骤如下:

(1) 按照"内容过滤"部分中的步骤来设置 meetup_data_type: 'group'。

(2) 转到 Kibana|Visualize|New(位于顶栏),选择数据表(Data table),选择索引模式为 meetup。

(3) 将其中的 Metrics 聚合方式设置为 Count,并将其标注为"Number of groups"。

(4) 在 Buckets 中添加 Split rows 的聚合,方式设置为 Terms,并将字段设置为 category_name.keyword。

(5) 由于要了解前 20 大类别,所以将 Size 设为 20,并将其标签设置为 Category Name (类别名称)。

(6) 单击应用更改,界面中将生成一个数据表,如下图所示。

Category Name	Number of Groups
tech	1,109
career/business	676
language/ethnic identity	583
socializing	475
outdoors/adventure	280
food/drink	212
new age/spirituality	202
health/wellbeing	176
sports/recreation	165
games	141

可以看到,tech(技术)是最受欢迎的类别,而在 5600 余个 Meetup 群组之中,tech 与大于 1000 的群组数量是对应的。

如果进一步分析国家中的类别,可以为字段 country.keyword 再添加 Split row 和 Terms 聚合。这样的设置将生成如下图所示的数据表。

Category	Country	Number of Groups
tech	in	456
tech	il	192
tech	ru	156
tech	cn	153
tech	us	106
career/business	us	203
career/business	in	202
career/business	cn	109
career/business	il	95
career/business	jp	47

前两个类别显示了一个惊人的事实。美国是一个 Meetup 用户数量最多的国家,但大多数聚会都不是有关技术的类别;印度对技术方面表现出了很高的兴趣。

甚至可为类别生成标签云,就像为主题所做的那样。这里只需遵循用于话题地图的相同步骤,将 meetup_data_type 设为 group,并将聚合字段设为 category_name.keyword。Meetup 群组分为 33 个类别,而我们只想了解前 20 个这样的类别。将 Size 设置为 20,然后单击应用更改,此时界面中将生成类别的标签云。

如数据表所示,tech(科技)类别的群组数量最多,这使得 tech 一词在图中最大。这表明技术是最热门的 Meetup 类别。此外,如果仔细观察,其实人们在技术(tech)和事业



(career)方面投入的关注更多。

12.3.9 热门话题

分析 Meetup 活动中的热门话题是一个很好的例子,因为这样可以展示出哪些流行语的呼声最大,并逐渐成为 Meetup 活动中的流行趋势。在分析热门话题这件事中,没有比标签云更好的可视化方式了。实现热门话题在标签云中的可视化可以遵循以下几个步骤:

(1) 不需要为此设置任何过滤器,因为主题仅与 Meetup 群组相关联。

(2) 转到 Kibana|Visualize|New(位于顶栏),选择 Tag cloud(标签云),选择索引模式为 Meetup。

(3) 将 Tag size(标签数量)指标(metric)设置为 Count。

(4) 对于 bucket,只有一个类型 Tags;选择这个类型,并设置聚合的方式为 Terms。

(5) 每一个主题都有一个 urlkey 作为唯一的标识符,这对我们来说也是一个很好的关键字。在字段列表中选择 topicsURLKeys.keyword。

(6) 将 size 设置为 30,这一设置将确保在标签云中获取前 30 个关键字。

(7) 单击应用更改,标签云可视化将以下图所示的样子出现。



可以看到，社交网络之类的关键词在 Meetup 活动中留下了良好的印象。随着 Facebook、Google+、Twitter、Instagram 等社交网络日益普及，企业家精神也是人们正在关注的一件事，那就是建立良性竞争。而且，人们正在谈论文化、灵性、自我完善、餐饮等诸多方面。标签云展示出了很多目前正在发生的事情，不是吗？

请注意标签云可视化内容左下角的“向上箭头图标”(up arrow icon)。此图标允许以表格的形式查看数据，其中会列出十大热门主题以及分页，如下图所示。

Topic	Number of Groups
social	875
socialnetwork	748
entrepreneurship	585
newintown	527
self-improvement	427
language	418
professional-networking	412
fun-times	406
startup-businesses	376
diningout	369

12.3.10 Meetup 活动场所地图

这个可视化将在地图上显示 Meetup 活动场所。由于拥有经度和纬度，因此可以使用 Kibana 在地图中绘制 Meetup 活动场所的统计情况。在执行这种可视化之前，为每个提到的国家的前 100 个城市运行 Logstash。通过以下步骤来实现该可视化：

(1) 按照“内容过滤”部分中的步骤来设置 meetup_data_type: 'venue'。

(2) 转到 Kibana|Visualize|New(位于顶栏)，选择 Tile map(瓦片地图)，选择索引模式为 meetup。

(3) 将其中的 metrics 聚合方式设置为 Count，并将其标注为“Number of venues”。

(4) 对于 buckets，这里仅提供了一种选项：Geo coordinates。

(5) 将聚合方式设为 Geohash，并将 Field 设为 geoip.location。

(6) 单击应用更改，将生成/更新出 Meetup 活动场所的统计情况。

在地图上可以看到，圆圈越大的位置，Meetup 活动场所就越多。还可以在 Options 选项卡中更改设置，将相同的地图转换为热力图。

(7) 在 Options 选项卡中，将地图类型设置为热力图，然后单击运行按钮。此时界面中将会生成 Meetup 活动场所热力图。

可以看到，颜色较深的地区表示活动场所数量更多，而更多的活动场所也可能代表该地区举办过较多的活动。

12.3.11 Meetup 活动地图

类似于活动场所在地图中的可视化，也可以看到 Meetup 活动的实际位置。这种可视化将在地图上展示人们聚会的地点。按照以下步骤实现这一可视化：

(1) 按照“内容过滤”部分中的步骤来设置 meetup_data_type：'meetup'。

(2) 转到 Kibana | Visualize | New(位于顶栏)，选择 Tile map(瓦片地图)，选择索引模式为 Meetup。

(3) 将其中的 metrics 聚合方式设置为 Count，并将其标注为“Number of meetups”。

(4) 对于 buckets，这里仅提供了一种选项，即 Geo coordinates。

(5) 将聚合方式设为 Geohash，并将 Field 设为 geoip. location。

(6) 单击应用更改，将生成/更新 Meetup 活动的统计情况，展示各个 Meetup 活动的举办地点。

类似于以上这些，也可以在地图上显示群组。只需要更改过滤器设置为 meetup_data_type：group，之后地图中将绘制出群组统计情况的可视化内容。

12.3.12 仅数量方面的统计

前面已经为各种例子都使用了可视化，事实上，最终还需要知道数据中具体有多少群组、Meetup 活动数量、活动的场所等，甚至是一些平均值。下面按照以下步骤设置指标实现可视化：

(1) 按照“内容过滤”(Filtering Content)部分中的步骤来设置 meetup_data_type：'meetup'。

(2) 转到 Kibana | Visualize | New(位于顶栏)，选择 Metric，选择索引模式为 Meetup。

(3) 将其中的 metrics 聚合方式设置为 count，并将其标注为“Number of meetups”。

(4) 单击运行按钮，将得到这些 Meetup 活动的具体数量。

(5) 将此可视化文件保存为 Number of Meetups Metric。

(6) 将 meetup_data_type 分别更改为 venue 和 group，可以按照相同的步骤对活动场所和群组执行可视化。当然，同时也应修改各自的标签。

(7) 完成后，可以将所有 metrics 统计数据添加到一个面板中，此时这些可视化内容已经就绪，如下图所示。

可以看到，有大约 5600 多个 Meetup 群组，在约 3100 多处地点组织了约 3600 多个活动。

12.4 获取通知

分析数据内容是有目的的，其中主要是出于战略性规划。如果这个分析是由一个处理人际关系的公司完成的，则可能有兴趣获取任何新的 Meetup 活动的信息，以便了解到更多的人。如果城市中的聚会能收到大量答复，那么这个公司能联系 3～4 个感兴趣的人物。这通常在 Meetup 活动期间完成，有相似兴趣的人都会走到一起。假如想在任何 Meetup 活动创建时收到某种通知，可以使用 X-Pack 的警报功能——Watcher 来实现。这里的通知可以是日志、电子邮件或者 Watcher 支持的其他一些类型。要了解有关 Watcher 的更多信息，请参阅第 10 章 X-Pack 插件中的 Alerting、Graph 和 Reporting 组件。

下面为这个用例创建一个 watch，需要 schedule（调度）、input（输入）、condition（条件）和 action（操作），用于发送通知：

- schedule：假设程序在一个小时内进行一次迭代，并对数据进行更新，将把调度程序的间隔设置为 1 小时。
- input：将使用一个查询，它会根据数据更新的时间（当前设置的时间是 1h），来检查一个城市中是否有新的 Meetup 活动，以及统计 yes RSVP（邀请成功）计数大于 100 的活动。
- condition：检查命中的数量是否大于零。
- action：使用电子邮件作为通知方式。

要发送电子邮件，应该将配置添加到 elasticsearch.yml。对于 Gmail，配置应如下所示：

```
xpack.notification.email.account:
    gmail_account:
        profile: gmail
        smtp:
            auth: true
            starttls.enable: true
            host: smtp.gmail.com
            port: 587
            user: <username>
            password: <password>
```

实际配置时，应该用实际值来替换用户名<username>和密码<password>。Watcher 将使用这些设置并使用 SMTP 服务来发送邮件。要了解有关电子邮件配置更多

信息请参阅 https://www.elastic.co/guide/en/x-pack/5.1/actions-email.html#configuring-email。

watch 应如下所示：

```
PUT _xpack/watcher/watch/meetup
{
    "trigger": {
        "schedule": {
            "interval": "1h"
        }
    },
    "input": {
        "search": {
            "request": {
                "indices": [ "meetup" ],
                "body": {
                    "size": 0,
                    "query": {
                        "bool": {
                            "filter": [
                                {
                                    "range": {
                                        "created": { "gte": "now-1h" }
                                    }
                                }
                            ]
                        }
                    }
                }
            }
        }
    },
    "condition": {
        "compare": {
            "ctx.payload.hits.total": {
                "gt": 0
            }
        }
    },
    "actions": {
        "email_me": {
```

```
                "throttle_period": "10m",
                "email": {
                    "from": "<from-email>",
                    "to": "<to-email>",
                    "subject": "Watchable Meetups",
                    "body": {
                        "html": "New Meetups within last hour - {{ctx.payload.hits.
                    total}}"
                    }
                }
            }
        }
    }
```

如前所述，将时间间隔设置为 1 小时。索引名称与 Meetup 的名称相同，在 watch 的输入部分中指定了一个查询，它会检查在一小时内创建的 Meetup 活动，设置的条件是至少有一个 Meetup 活动被创建。如果条件匹配，那么程序将发送电子邮件通知。在 from 部分的电子邮件地址与 elasticsearch.yml 配置中提到的相同，to 字段中的电子邮件地址是通知要发送的地址。

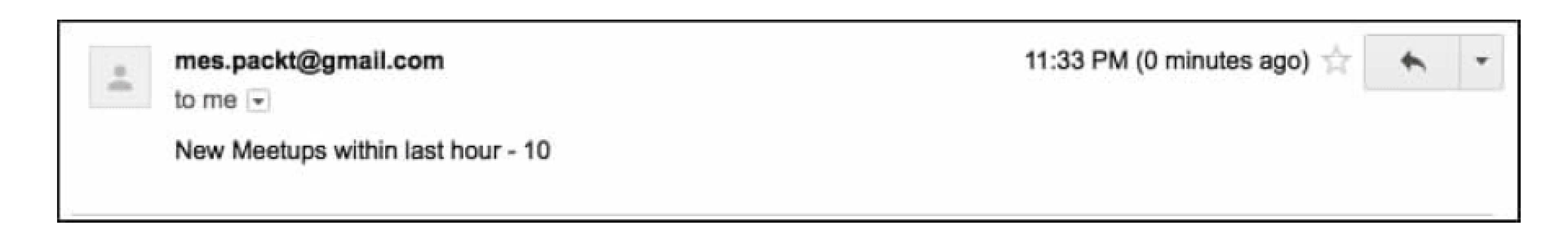
可以使用以下方式来注册 watch：

```
PUT _xpack/watcher/watch/meetup_watch
{
    ...watch body ...
}
```

在 watch 主体部分应由上面列表中的 watch 取代，它们将按时执行。要手动执行一次，可以执行以下命令：

```
POST _xpack watcher/watch/meetup_watch/_execute
```

如果在过去一小时内有任何 Meetup 活动被创建，就会收到一封如下图所示的电子邮件：



邮件中提到，在过去一小时内创建了 10 个新的 Meetup 活动。

这是一个非常简单的 watch 示例，它执行了发送电子邮件的操作。可以添加更多的过滤器来检查是否在特定城市中创建了 Meetup 活动，或者成功邀请计数超过 100 个。

12.5　本章小结

Meetup 活动是一个很好的分析案例。事实上，我们使用了多种可视化内容来了解 Meetup 活动的情况，了解了现有插件为什么不能满足我们的需求，还有定制的新插件带来了什么新功能。我们通常对问题的探究总是很感兴趣，对于这里提到的 Meetup，目前是一个很流行的平台，它有助于传播知识和善行。我们了解了如何使用瓦片地图(tile map)来查看大多数 Meetup 活动举办的地区；标签云帮助我们了解了大量 Meetup 活动的热门话题和类别。最后，我们看到了 Watcher 如何帮助我们收到新创建活动的通知。

Elastic Stack 提供的不仅是我们在书中涵盖的内容。该工具集至今仍在发展，今后也将提供更多值得期待的组件和功能。我们希望书中涵盖的内容和案例能够做到准确无误，并对你有所帮助，也希望这本书可以帮你学习如何有效地利用 Elastic Stack 来解决实际使用中的问题。每个使用案例都需要一些特别的东西，有时可能需要新的插件、新的可视化，或者完全不同寻常的数据分析方法。要根据面对的实际使用场景，做出不同的决定。

本书的所有代码文件可以在 Github 存储库上的分支 5.1.1 中找到，详见 https://github.com/kravigupta/mastering-elastic-stack-code-files/tree/5.1.1。类似地，同一存储库的 Wiki 页面上提供了其他相关和有用的内容或注释，详见 https://github.com/kravigupta/mastering-elastic-stack-code-files/wiki。

接下来是什么？好吧，现在故事刚刚开始。每个开源软件都需要贡献者的支持社区，任何活跃的社区在学习和实现中都扮演着重要的角色。Elastic 社区 https://www.elastic.co/community 与维护良好的论坛 https://discuss.elastic.co/都非常活跃。可以在社区中注册，并分享你的知识和学习心得。Elastic 文档和社区都会发布最新动态，每个组件都有一些 IRC 频道，托管在 http://freenode.net 网站中。这些 IRC 频道如下：

- Elasticsearch：https://webchat.freenode.net/#elasticsearch
- Logstash：https://webchat.freenode.net/#logstash
- Kibana：https://webchat.freenode.net/#kibana
- Beats：https://webchat.freenode.net/#beats

这些频道都相当活跃，欢迎加入到这个群体中来。合作快乐！学习快乐！